SORBONNE UNIVERSITÉ

ÉCOLE DOCTORALE V : Concepts et Langages

UR 4509 : Sens, Texte, Informatique, Histoire (STIH)

T H È S E

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ SORBONNE UNIVERSITÉ

Discipline : Linguistique

Présentée et soutenue par :

Yoann AMAR ASSOULINE

le : 1er juillet 2024

Approche sémio-informatique du Développement Logiciel Tridimensionnel :

Programmation, interthéoricité et interactivité vidéoludique

Sous la direction de :

Mme Astrid GUILLAUME – Maîtresse de conférences HDR en Sciences du langage, Sorbonne Université – Faculté des lettres

Membres du jury :

M. Thomas BRODEN, Professeur émérite des Universités, Purdue University, États-Unis

Mme Valérie BRUNETIÈRE (Présidente), Professeure des Universités, Université Paris Cité

M. Didier TSALA-EFFA, Professeur des Universités, Université de Limoges

M. Juan ALONSO-ALDAMA, Maître de conférences HDR, Université Paris Cité

Page intentionnellement laissée vide.

SOMMAIRE

Pages liminaires IV

Remerciements V

Conventions de la thèse VI

Liste des acronymes et sigles XI

1. Introduction : Genèse de l'interactivité.....1

1-1. Contextualisation multifacette 2

1-2. Scientifisation des jeux vidéo..... 42

1-3. Structure globale de la thèse 74

2. Développement Logiciel Tridimensionnel : Spécifications 82

2-1. Passerelle technologique infrangible 83

2-2. Générateur de théories vidéoludiques.....107

2-3. Synthèse du DLT 119

3. Cinquième génération : Prémices des moteurs 3D 124

3-1. Assemblage des premiers jeux 125

3-2. Sémiotisation interactive des univers 3D..... 157

3-3. Synthèse de la cinquième génération 184

4. Sixième génération : Incrémentation des moteurs 3D.....188

4-1. Technologisation vidéoludique 189

4-2. Caractères primordiaux interactifs.....214

4-3. Synthèse de la sixième génération..... 233

5. Septième génération : Polymorphie des moteurs 3D 238

5-1. Super-ordinateurs de divertissement 239

5-2. Interactants technologiques 252

5-3. Synthèse de la septième génération..... 269

6. Huitième génération : Hyperstructure des moteurs 3D.....272

6-1. Standardisation des technologies 273

6-2. Hyperstructure 3D 284
6-3. Synthèse de la huitième génération 296

7. Conclusion : Immuabilité technologique 298

7-1. Technologisation multidimensionnelle 299
7-2. Standardisation industrielle protéiforme 307
7-3. Infinité des univers quantiques 315

Références 318

Bibliographie 319
Ludographie 341
Médiagraphie 345
Liste des illustrations 356

Pages annexes 358

Annexe A : Schémas originaux 359
Annexe B : Nouveau corpus vidéoludique 360
Annexe C : Présence des genres vidéoludiques 362
Annexe D : Corpus développemental indirect 366
Annexe E : Ancien corpus vidéoludique 368

Glossaire 381

Table des matières 384

PAGES LIMINAIRES

REMERCIEMENTS

Je voudrais tout d'abord exprimer ma profonde gratitude envers ma directrice de recherche, Madame Astrid GUILLAUME, pour sa bienveillance à mon égard, son écoute et sa disponibilité sans faille. Elle m'a fait l'honneur d'avoir accepté de diriger cette thèse doctorale pendant plusieurs années, et ses qualités humaines et académiques exceptionnelles m'ont aidé à progresser dans cette recherche et à atteindre le résultat actuel.

Ensuite, je tiens à remercier sincèrement Madame Valérie BRUNETIÈRE et Monsieur Didier TSALA-EFFA d'avoir accepté de rédiger les pré-rapports de ma thèse doctorale. Ce fut un véritable privilège d'avoir bénéficié de multiples commentaires d'une grande pertinence sur ce conséquent travail, me permettant de pouvoir démarrer la soutenance sereinement.

Je souhaite également remercier l'ensemble des membres du jury, la présidente Madame Valérie BRUNETIÈRE, Monsieur Juan ALONSO-ALDAMA, Monsieur Thomas BRODEN et enfin Monsieur Didier TSALA-EFFA, qui ont bien voulu m'honorer de leur présence dans le cadre de ma soutenance. Cela a permis une évaluation très professionnelle et pertinente de mon travail. Je les remercie grandement de leur rigueur et de leur sérieux.

J'aimerais également adresser mes plus cordiaux remerciements à l'égard de tous les membres du personnel de Sorbonne Université, qui m'ont permis de réaliser ma recherche sous des conditions idéales et de réaliser sereinement les différentes démarches administratives.

Plus encore, je souhaite remercier chaleureusement toutes les personnes de mon entourage, qu'elles soient proches ou éloignées, qui m'ont apporté un gargantuesque soutien et des encouragements dans la réalisation de mes projets.

Je tiens enfin à exprimer, et cela du plus profond de mon cœur, toute ma gratitude pour le soutien indéfectible et infrangible de ma mère. Depuis les premiers pas de cette recherche et jusqu'à présent, elle a toujours su être là pour moi, m'apportant son amour et sa joie de vivre dans tous ces moments.

CONVENTIONS DE LA THÈSE

Dans l'optique de mieux répondre aux multiples enjeux de notre recherche, il nous semble nécessaire d'établir les fondations des conventions que nous utiliserons dans l'ensemble de notre manuscrit. L'intérêt principal étant, d'une part, de respecter les normes imposées par un tel travail universitaire et, d'autre part, de faciliter substantiellement l'écriture ainsi que la lecture de cette recherche. Ces conventions indiquent notamment le style bibliographique utilisé et les multiples modifications apportées à celui-ci, mais également les normes typographiques de chaque troisième section.

Ainsi, le style bibliographique employé dans cette thèse doctorale est nommé *Chicago 17th B*, et il s'agit plus précisément de la variante *auteur-date* (ou *author-date*), dont le guide officiel est accessible sur le site du *Chicago Manual of Style*[1]. Nous avons également constaté qu'il n'y a aucune obligation officielle pour l'usage d'un style bibliographique particulier à Sorbonne Université. De surcroît, nous avons observé l'usage du style Chicago *17th B* abondamment utilisé dans les études traitant des jeux vidéo, et c'est pourquoi nous avons considéré que ce même style était particulièrement pertinent pour notre recherche. Néanmoins, afin d'harmoniser cette bibliographie, et à cause du fait qu'il « n'existe pas de traduction française officielle du Chicago Manual of Style » (Université de Montréal. s. d.), nous décidons d'utiliser la traduction proposée par l'Université de Montréal[2]. Par ailleurs, et à l'aune des besoins de notre thèse, nous prenons la liberté de modifier quelques aspects de ce style bibliographique, en exposant d'abord plusieurs modifications primaires ci-dessous.

La première concerne la volonté de fournir une traduction intégrale des citations en anglais dans le corps de la thèse. Cette traduction est systématiquement effectuée par moi-même, l'auteur de la thèse, tout en restituant la citation d'origine dans une note infrapaginale.

La deuxième, toujours liée aux citations du corps de la thèse, permet de clarifier les citations provenant de documents électroniques n'ayant aucun

1. *The Chicago Manual of Style Online*. https://www.chicagomanualofstyle.org.
2. Université de Montréal. s. d. « Citer selon le style Chicago: À propos ». Université de Montréal. https://bib.umontreal.ca/citer/styles-bibliographiques/chicago.

système de pagination. Celle-ci implique de mentionner, lorsque le document consulté est structuré, la section ou le chapitre d'où provient la citation. Nous pouvons illustrer cette modification via cet exemple, pour une citation située dans le chapitre 24 d'un ouvrage consulté : (Hansen 2016, ch. 24).

La troisième modification est liée aux documentations informatiques qui ne possèdent aucune pagination classique. Ainsi, lorsque nous mentionnons une documentation particulière, par exemple celle du moteur *Unreal Engine 4*, alors nous utiliserons le nom de la partie concernée. Exemple avec la section nommée *What's New* de la documentation du moteur : (Epic Games 2021, What's New?).

La quatrième modification concerne la bibliographie, dans laquelle le nombre total de pages est systématiquement ajouté à la fin de la référence complète des documents concernés. On l'illustre avec cet exemple : 320 pour une lecture complète ou 200-220 pour la consultation d'un chapitre ou d'une section d'un livre.

La cinquième modification est également liée à la bibliographie, où nous réécrivons intégralement les noms et prénoms des multiples auteurs, même lorsque nous utilisons différentes références d'un seul et même auteur. Nous omettons ainsi les trois tirets cadratins propres au style bibliographique du *Chicago 17th B* (———). Par exemple, lorsque nous avons plusieurs entrées dans notre bibliographie pour Steven L. Kent, cela implique dans notre cas de réécrire le nom et prénom de cet auteur à chaque nouvelle entrée.

Au-delà de ces modifications primaires, et au regard des outils informatiques et des jeux vidéo présents dans notre recherche, il existe des conventions proposées par le style bibliographique *Chicago 17th B* pour justement citer ces mêmes logiciels ou jeux vidéo. Cependant, elles ne semblent pas assez précises, aussi bien pour notre problématique que pour respecter notre volonté de rigueur dans nos analyses. C'est pourquoi nous proposons deux systèmes distincts que nous décrivons ci-dessous, un premier pour les outils informatiques et un second pour les œuvres vidéoludiques.

NOMMAGE DES OUTILS INFORMATIQUES

Dans cette recherche, nous analysons de multiples outils informatiques, notamment des langages de programmation, moteurs de jeu ou logiciels divers, de deux manières distinctes : avec ou sans mention de la version de l'outil

concerné. Ainsi, lorsque nous évoquons un outil en omettant sa version, cela signifie que nous analysons l'outil en général et ses caractéristiques globales. Par exemple, nous citerons le langage de programmation *C*, en général, pour analyser ses caractéristiques et propriétés intrinsèques.

Cependant, la mention d'une version précise d'un outil informatique signifie que nous souhaitons mettre en exergue une caractéristique liée à la version en question. Ceci parce que les caractéristiques de chaque version, majeure ou mineure, peuvent être substantiellement différentes, parfois au point d'avoir des incompatibilités. Cela peut s'illustrer à travers un exemple du chapitre 2, où nous évoquerons la version *C89* ou *C90* pour répondre à des points précis de nos analyses, qui ne concernent plus le langage en général, mais plutôt une version particulière de celui-ci. Précisons également que nous n'utilisons pas de système de citation scientifique pour les outils informatiques, car nous estimons que la version, ou l'absence de version, est suffisante pour la rigueur de nos analyses. De surcroît, dans un tel cas, aucune confusion n'est possible pour la citation d'un outil informatique.

CITATIONS DES ŒUVRES VIDÉOLUDIQUES

Au regard des jeux vidéo, nous estimons qu'ils ont également besoin d'un système de citation robuste et précis pour notre recherche, ce que nous pouvons illustrer via un jeu vidéo spécifique : *Metal Gear Solid 3*. On pourrait d'abord être tenté de citer cette œuvre, avec ce titre générique, tout en précisant le développeur et l'année. Or, il existe plusieurs versions du jeu, ayant des différences substantielles :

- *Metal Gear Solid 3: Snake Eater* : la version originale du jeu, publiée sur la console PlayStation 2 entre 2004 et 2005,
- *Metal Gear Solid 3: Subsistence* : une version améliorée, agrémentée de missions supplémentaires, publiée sur les consoles *Xbox* et *PlayStation 2* ainsi que sur *PC* entre 2005 et 2006,
- *Metal Gear Solid 3 HD Edition* : une version *HD Remaster* possédant des missions supplémentaires (mais omettant le mini-jeu de skateboard), et publiée sur les consoles *PlayStation 3*, *Xbox 360* et *PlayStation Vita* entre 20011 et 2012.

- *Metal Gear Solid 3D: Snake Eater* : une version portable publiée sur la *Nintendo 3DS* en 2012.
- *Metal Gear Solid Δ: Snake Eater* : un remake en cours de développement et qui sera publiée en 2024.

On peut constater qu'il existe, en réalité, au moins cinq versions différentes du même jeu, et une citation sans préciser la version peut potentiellement rendre l'analyse problématique. De surcroît, les consoles *PlayStation 3* et *Xbox 360* sont rétrocompatibles, du moins pour les premières versions de la *PlayStation 3*, donc la citation du matériel utilisé ne pourrait éluder l'ensemble des problèmes. C'est pourquoi on précisera le *titre complet*, et cela lors de chaque occurrence d'un jeu vidéo, où on indiquera toutes les informations pouvant désambiguïser notre propos et cibler, au mieux, la version concernée. On ajoutera également la première console ou plate-forme où le jeu en question a été distribué. On utilisera en outre la mention *Multi* si un même jeu a été publié sur plusieurs consoles simultanément.

Toujours pour les citations, aucun usage des locutions latines *ibidem* ne sera fait lorsque nous citons les jeux vidéo, locutions que nous réservons strictement à des auteurs de la bibliographie, afin de ne pas générer de potentielles confusions. Enfin, précisons pour les citations d'œuvres vidéoludiques que, lorsque nous citons une série entière et non pas un jeu précis, nous utilisons la préposition latine *ab*, puis omettons l'éditeur et ajoutons uniquement la première console ou plate-forme de publication.

Citation	
	Titre d'un seul jeu (Développeur Année de publication, Console).
	Grand Theft Auto III (Rockstar Games 2001, PS2).
	Titre d'une série (ab. Première année de publication, Première console).
	Grand Theft Auto (ab. 1997, PS1*).*

Tableau 1. Conventions des citations de la ludographie

Référencement des œuvres (ludographie)

Afin de préciser certains éléments au regard de la ludographie présente dans le chapitre *Références* de notre thèse, signalons d'emblée que certains jeux vidéo possèdent plusieurs noms en fonction des continents, ou de ce qu'on peut appeler des *régions* dans l'industrie vidéoludique. Dans un tel cas de figure, nous

utilisons exclusivement l'appellation américaine des différents jeux. Par exemple, le jeu *Driver* est nommé *Driver: You Are the Wheelman* aux États-Unis, et c'est ce dernier nom américain qui sera repris dans la thèse. Nous respectons et appliquons également les règles de ponctuation de l'anglais américain, notamment l'absence d'espace insécable entre un mot et un signe de ponctuation comme le point d'interrogation, d'exclamation ou encore le deux-points.

Enfin, s'il existe plusieurs éditeurs pour un même jeu, alors nous citons uniquement le premier éditeur américain, ou japonais s'il n'y en a pas, voire européen s'il n'y a que des éditeurs européens. Si l'éditeur a changé de nom, nous utiliserons le nom le plus récent, afin de conserver une consistance et une harmonie pour l'ensemble des jeux, sauf dans les cas où cela peut entraîner une confusion dans la lecture de la recherche.

LUDOGRAPHIE	
	Titre du jeu. (Première année de publication) Dernière année de publication. Développeur. Éditeur. Liste des plates-formes par ordre alphabétique remplaçable par *et al* pour plus de 5 plates-formes.
	Grand Theft Auto III. (2001) 2021. Rockstar North. Rockstar Games. PlayStation 2 et *al*.

Tableau 2. Conventions des références de la ludographie

TROISIÈME SECTION DES CHAPITRES

Au regard de la dernière précision des conventions de la thèse, nous pouvons indiquer que, dans notre manuscrit, chaque chapitre est divisé en trois sections. Et la troisième section est généralement réservée à la synthèse ou le récapitulatif du chapitre entier, à l'exception de l'introduction et de la conclusion. Surtout, cette troisième section possède une spécificité typographique : chaque titre de chapitre, de section ou de sous-section est indiqué en **gras**. L'intérêt d'une telle mise en forme est de créer des indicateurs visuels plus limpides au regard de ce que chaque paragraphe synthétise, afin de mieux assimiler les informations mais également de pouvoir lire plus rapidement cette section synthétique.

LISTE DES ACRONYMES ET SIGLES

API : *Application Programming Interface*

CLG : Cours de Linguistique Générale

CPU : *Central Processing Unit*

DCC : Digital Content Creation (logiciel)

Famicom : Family Computer

FBX : Filmbox

FPS : *First-person Shooter* ou *Frame Per Second*

GDD : *Game Design Documents*

GPU : *Graphics Processing Unit*

GTA : Grand Theft Auto

HAL : Hardware Abstraction Layer

HD : *High-Definition*

MGS : *Metal Gear Solid*

MMORPG : *Massively Multiplayer Online Role-Playing Games*

MOBA : *Multiplayer Online Battle Arena*

NDT : Note du traducteur

NES : *Nintendo Entertainment System*

NPC : *Non-Player Character* ou *Non-Playable Character*

PC : *Personal Computer* ou *Player Character* ou encore *Playable Character*

SDK : Software Development Kit

TDR : Traduction du rédacteur

TN : *Translator's Note* (note du traducteur)

UDK : Unreal Development Kit (Unreal Engine 3 Public Version)

UE : Unreal Engine

WYSIWYG : *What You See Is What You Get*

1. INTRODUCTION :

GENÈSE DE L'INTERACTIVITÉ

1-1. CONTEXTUALISATION MULTIFACETTE

Depuis les prémices du XXIe siècle, le développement de jeux vidéo en 3D temps réel implique l'implémentation d'une kyrielle de mécanismes interactifs dans un seul et même univers virtuel, un phénomène qui semble standardiser une liberté de l'interactivité potentiellement infinie. Communément nommées *gameplay* ou jouabilité, ces mêmes interactions comprennent, d'après Hermès, « l'ensemble (...) [des] caractéristiques ludiques, [c'est-à-dire] les règles d'un jeu, sa prise en main, l'univers dans lequel il se déroule [et] l'expérience vécue par le joueur » (Hermès 2012, 15). En d'autres termes, tous les éléments qui favorisent une expérience de jeu unique. Cette hypothétique liberté du *gameplay* dans les œuvres vidéoludiques semble parfois être en corrélation avec une autre composante fondamentale : le monde ouvert. Concept relativement récent, celui-ci offre, comme souligné par Gillian Dale et C. Shawn Green, aux « joueurs (…) une grande liberté pour choisir où aller, quels objectifs poursuivre et comment aborder les différents événements qui surgissent » (Dale et Shawn Green 2017 TDR)[3].

De surcroît, c'est surtout *Grand Theft Auto III* (Rockstar Games 2001, PS2) qui a largement démocratisé ce modèle au début des années 2000. Un jeu considéré comme étant, selon Dustin Hansen, un véritable « précurseur du genre [de jeux] à récit non linéaire [où] les joueurs peuvent se balader librement dans le monde ouvert de Liberty City, et faire avancer l'histoire à leur propre rythme » (Hansen 2016, ch. 24 TDR)[4]. Cependant, bien avant ce jeu de *Rockstar Games*, il existait déjà des œuvres vidéoludiques avant-gardistes, qui développaient une forme prototypique d'un monde ouvert en 3D. Nous pouvons citer plusieurs exemples, notamment *Driver: You Are the Wheelman* (Atari 1999, PS1) ou encore *Body Harvest* (DMA Design 1998, N64). Ce dernier est habituellement perçu, d'après les informations de Matt Barton, comme étant « un titre 3D innovant qui

3 « players (…) a great deal of freedom to choose where to go, which objectives to take on, and how to approach the various events that arise » (Dale et Shawn Green 2017).
Rappel : le sigle TDR signifie *Traduction du rédacteur*.

4 « pioneered the nonlinear story-driven [games] genre [where] players are allowed to cruise through the open world of Liberty City and move the story along at their own pace » (Hansen 2016, ch. 24).

présageait bien ce qui allait arriver avec *GTA III* » (Barton 2019, 303 TDR)[5]. Et si Barton établit un lien entre les deux titres en question, il ne s'agit nullement d'une coïncidence. En effet, *Body Harvest* (DMA Design 1998, N64) est en réalité développé par la même équipe que la saga *Grand Theft Auto* (*ab.* 1997, PS1), puisque le « premier GTA fut [également] créé par DMA Design » (Barton 2019, 303 TDR)[6]. Par la suite, ce studio sera connu sous un autre nom, si nous reprenons les propos de Jonathan Bolding, notamment depuis qu'il « a été acquis par Rockstar en 1999 et renommé Rockstar North » (Bolding 2022 TDR)[7].

Néanmoins, pour esquisser une perspective historique plus complète du concept de monde ouvert, on peut mettre en relief le jeu supposé précurseur de cette même liberté virtuelle. Il s'agit notamment, selon le descriptif d'Andrew Williams, du « simulateur de vol spatial *Elite* (1984, Acornsoft Limited) [qui] avait l'un des plus vastes mondes de jeu créés (...) [et était] l'une des premières occurrences commerciales d'un *gameplay* bac à sable en monde ouvert » (Williams 2017, 126-127 TDR)[8]. De surcroît, et comme le souligne Barton, ce simulateur a joué un rôle substantiel dans la conception de la saga *Grand Theft Auto* (*ab.* 1997, PS1), puisque « Gary Penn, un des développeurs du jeu [*GTA*], cite *Elite* (...) comme étant une influence majeure, allant jusqu'à appeler GTA "Elite dans une ville" » (Barton 2019, 303 TDR)[9]. On peut également citer un autre exemple datant de la même époque, bien plus profondément ancré dans l'imaginaire collectif : le premier *The Legend of Zelda* (Nintendo 1986, NES). En effet, ce dernier est souvent considéré comme étant un authentique pionnier pour Hansen, dans lequel, « pour la première fois, les joueurs ont été envoyés dans de longues aventures en monde ouvert » (Hansen 2016, ch. 8 TDR)[10].

Nous mettons néanmoins un terme à ces considérations quasiment éristiques de la chronologie d'apparition des jeux en monde ouvert car, avant de naviguer

5. « an innovative 3D title that foreshadowed much of what was to come with *GTA III* » (Barton 2019, 303).
6. « first GTA was [also] created by DMA Design » (Barton 2019, 303).
7. « was acquired by Rockstar in 1999 and renamed Rockstar North » (Bolding 2022).
8. « space-based flight simulator *Elite* (1984, Acornsoft Limited) [which] had one of the largest game worlds created (...) [and was] one of the earliest commercial instances of open world sandbox gameplay » (Williams 2017, 126-127).
9. « Gary Penn, one of the [GTA] game's developers, cites *Elite* (...) as a key influence, going so far as to call GTA "Elite in a city" » (Barton 2019, 303).
10. « for the first time, players were sent on long, open-world adventures » (Hansen 2016, ch. 8).

dans les méandres de cette recherche, il nous paraît essentiel de présenter succinctement deux aspects majeurs : les objectifs et caractéristiques de notre thèse, d'une part, mais également un historique relativement succinct de la naissance des jeux vidéo, d'autre part. Cette double recontextualisation permet de mieux appréhender l'origine de notre problématique, fortement liée à l'interactivité et représentant naturellement le fil d'Ariane de notre étude. Et, après avoir présenté ces deux facettes de notre contextualisation, nous présenterons dans les sous-sections suivantes l'émergence de la taxinomie des genres vidéoludiques, puis nous enchaînerons sur la problématique centrale de notre recherche. Enfin, nous exposerons les deux corpus majeurs, à savoir le corpus développemental et le corpus vidéoludique.

Toutefois, précisons qu'au regard du contexte multifactoriel et protéiforme d'émergence des premiers jeux vidéo, nous décidons d'éluder l'histoire générale de l'informatique. En effet, plonger dans les arcanes de l'historique des premiers ordinateurs risquerait, à cause de son aspect profondément hétéroclite et éminemment complexe, d'orner notre discours de précisions superfétatoires et prolixes. Pire encore, on éloignerait *in fine* notre thèse du prisme initial de recherche, en l'occurrence de la sémiotique appliquée au développement de jeux vidéo, et non pas à l'histoire de l'informatique.

Ainsi, cette recherche doctorale en sémiotique appliquée développe une approche sémio-informatique inédite, nommée Développement Logiciel Tridimensionnel, et reliant informatique et sémiologie en interdisciplinarité. Celle-ci s'appuie principalement sur le concept d'interthéoricité d'Astrid Guillaume (2014), représentant *in fine* la clef de voûte de notre étude et de notre approche, permettant de surcroît de mieux réadapter nos multiples théories qui en découlent. Surtout, l'informatique est incorporée via l'examen méticuleux d'outils de développement de jeux vidéo 3D, principalement des langages de programmation et moteurs de jeu. Quant à la sémiologie, discipline intégrable en sciences du langage ou linguistique (ou inversement, selon les auteurs), elle permet de transcender nos analyses pour élaborer *in fine* des théories et structures *ex nihilo* ou *ex materia*. Plus proche d'une sémiotique appliquée que d'une sémiologie théorique, on s'évertuera à développer nos propres concepts, en s'inspirant de nombreuses théories provenant de Ferdinand de Saussure,

Algirdas Julien Greimas, Louis Trolle Hjelmslev, Jacques Fontanille, François Rastier ou encore Louis Hébert.

Plus encore, l'objectif primordial de cette approche novatrice est d'étudier l'existence, voire l'influence des genres vidéoludiques dans la création des multiples mécanismes interactifs, autrement dit du *gameplay*, au sein des outils de développement de jeux vidéo en 3D temps réel contemporains, et ce de manière diachronique, à travers la segmentation temporelle des générations de consoles, symbolisant de véritables paliers technologiques. Cette périodisation débute dès la genèse de la 3D temps réel, située au début des années 1990 avec la cinquième génération, et se termine à la fin de l'année 2019, en clôturant les analyses sur la huitième génération. Nous omettons ainsi la neuvième génération car celle-ci est toujours en cours d'évolution lors de la rédaction de cette recherche, et nous souhaitons éviter les spéculations ou encore les imprécisions induites par une telle situation.

Afin de réaliser cette étude scientifique, nous nous appuyons essentiellement sur deux corpus distincts : des outils de développement de jeux vidéo 3D contemporains et des données discursives sur le développement de jeux vidéo commerciaux. Surtout, cette division bicéphale de l'analyse permettra *in fine* de concrétiser un autre dessein esquissé par cette recherche : ériger une passerelle infrangible entre l'industrie professionnelle du développement de jeux vidéo 3D et des théorisations scientifiques sur l'interactivité. Cette même passerelle permettra de générer certes de nombreuses théories, mais surtout une hyperstructure 3D, c'est-à-dire une structure sémiotique et informatique inédite, qui va mettre au jour les principes de la création d'univers virtuels.

Néanmoins, s'aventurer dans l'histoire sibylline des œuvres vidéoludiques est un processus laborieux, impliquant *ipso facto* de sélectionner avec acuité les jeux vidéo majeurs du temps jadis. Des œuvres subsumées par une représentation simplificatrice et une interactivité élémentaire, ayant pourtant fortement contribué à la croissance d'une industrie encore naissante. Or, nonobstant notre volonté d'exactitude et de rigueur dans nos propos, nous renonçons à l'idée d'établir un panorama exhaustif d'un paysage vidéoludique si opulent. En effet, les quelques milliers de jeux vidéo qui accompagnent inéluctablement chaque console de jeu complexifient substantiellement le maintien d'un équilibre entre historicité et exhaustivité. À titre indicatif, la boutique en ligne de jeux vidéo

digitalisés *Steam* répertorie, en 2022 et selon Dustin Bailey, un « nombre total de jeux (...) [qui] a dépassé les 50 000 » (Bailey 2022, TDR)[11]. C'est pourquoi il sera nécessaire, tout au long de cette thèse, d'ignorer inéluctablement une grande partie des œuvres vidéoludiques, mais également de nombreux outils de conception de jeux vidéo. Néanmoins, nous sélectionnerons les œuvres et outils les plus représentatifs et les plus marquants de l'industrie vidéoludique.

Quoi qu'il en soit, l'origine des jeux vidéo semble provenir, pour Steven Kent, d'une « relation étroite avec l'industrie du divertissement déjà bien établie » (Kent 2001, 2 TDR)[12]. Plus précisément, il indique qu'elle est intrinsèquement liée au « commencement du flipper [qui] peut être attribuable au Bagatelle, une variante du billard » (*ibid*. TDR)[13]. Aujourd'hui tombé en désuétude, hypothétiquement en raison de la prédominance des jeux vidéo, on peut constater que « les premiers flippers à pièces ont été introduits par David Gottlieb en 1931 » (Barton 2019, 1 TDR)[14]. Ainsi, elles s'installaient insidieusement dans le divertissement quotidien, peu après le début de la Grande Dépression. Au fil des années, on constate surtout, toujours avec Barton, que les « flippers et jeux similaires avaient déjà établi un marché et habitué les gens à l'idée de salles d'arcade, de jeux (...) et de la culture vidéoludique » (*ibid*. TDR)[15]. Ce qui, en d'autres termes, avait développé un terrain véritablement propice à l'émergence des jeux vidéo.

Parallèlement, et avant les premières machines vidéoludiques fonctionnelles, on observait dès la fin des années 1940 des schématisations purement théoriques du jeu vidéo. L'un de ces concepts était mentionné, selon Djaouti, dans un « brevet américain, déposé le 25 janvier 1947 et accordé le 14 décembre 1948 pour une machine nommée le **Cathode Ray Tube Amusement Device** (...) [des] concepteurs (...) **Thomas Toliver Goldsmith Jr.** et **Estle Ray Mann** » (Djaouti 2019, 13). Une idée qui paraissait probablement révolutionnaire à l'époque,

11 « total number of games (...) [which] has surpassed 50, 000 » (Bailey 2022).
12. « close relationship with the well-established amusement industry » (Kent 2001, 2).
13. « beginnings of pinball [which] can be traced back to Bagatelle, a form of billiards » (Kent 2001, 2).
14. « the first coin-operated pinball machines were introduced by David Gottlieb in 1931 » (Barton 2019, 1).
15. « pinball and similar games had already established markets and accustomed people to the idea of arcades, (...) games, and gaming culture » (Barton 2019, 1).

d'après Djaouti, notamment grâce à sa forme d'interactivité prototypique, où « le joueur doit guider un missile, représenté par un point se déplaçant à l'écran, vers des cibles » (*ibid.*). Nonobstant son ancienneté, Djaouti indique qu'il n'existe toujours « aucune trace de la construction d'une telle machine » (*ibid.*).

Peu après, l'ingénieur Ralph Henry Baer avait également, « depuis l'année de 1950-1951 (...) à Loral Electronics Corp (...) suggéré à (...) l'ingénieur en chef (..) [de] fabriquer une certaine forme de jeu dans (...) [un] téléviseur » (Baer 2005, 3 TDR)[16]. Malgré son aspect avant-gardiste, il a indiqué que l'idée est restée inexploitée, puisque la « réponse (...) était (...) oublie cela [et] fabrique seulement le (...) téléviseur » (*ibid.* TDR)[17]. Par ailleurs, on peut également citer Stuart Brown, qui indique que le « premier jeu vidéo [est], (...) *Draughts* de Christopher Strachey (...) dont le fonctionnement est confirmé (...) [en] juillet 1952 » (Brown 2019, 50:38-50:49 TDR)[18]. Peu populaire, cet exemple semble être habituellement absent de la littérature vidéoludique.

Néanmoins, les premiers jeux ayant bénéficié d'une médiatisation significative ont plutôt vu le jour dans de multiples laboratoires, à l'instar de *Tennis for Two* (Higinbotham 1958*,* Donner Model 30), souvent cité, par exemple par Daniel Ichbiah, comme étant le « premier jeu électronique de l'Histoire » (Ichbiah, 2014). Celui-ci fut élaboré, pour reprendre les propos d'Andy Bossom et de Ben Dunning, « en 1958 (...) dans le Laboratoire National de Brookhaven (...) [par le] physicien nucléaire William Higinbotham » (Bossom et Dunning 2015, 20 TDR)[19]. Il dépeignait un concept calqué, d'après Williams, sur un « match de tennis, (...) [et utilisant un] oscilloscope pour l'affichage » (Williams 2017, 35 TDR)[20]. Très peu de temps après, un autre jeu tout aussi significatif émergea : *Spacewar!* (Russell 1962, PDP-1). Son développement avait commencé, selon Ichbiah, dès « 1961, (...) par le programmeur Steve Russel qui se trouvait au Massachusetts Institute

16. « ever since the year of 1950-1951 (...) at Loral Electronics Corp (...) suggested to Sam Lackoff (...) the Chief Engineer (...) [to] build some form of game into (...) [a] TV set » (Baer 2005, 3).
17. « answer was (...) forget it [and] just build the (...) TV set » (Baer 2005, 3).
18. « first video game [is] (...) Christopher Strachey's Draughts (...) confirmed working (...) [in] July 1952 » (Brown 2019, 50:38-50:49).
19. « in 1958 (...) at the Brookhaven National Laboratory (...) [by the] nuclear physicist William Higinbotham » (Bossom et Dunning 2015, 20).
20. « tennis match, (...) [and using an] oscilloscope for a display » (Williams 2017, 35).

of Technology, (...) afin de tester les capacités graphiques de DEC PDP-1 » (Ichbiah, 2014).

L'idée principale était, comme souligné par Bossom et Dunning, de pouvoir concevoir un « jeu de tir de vaisseau spatial à deux joueurs » (Bossom et Dunning 2015, 20 TDR)[21]. Surtout, c'est lors de la création de ce même jeu qu'on a observé les prémices de la théorisation de l'interactivité virtuelle. C'est ce que constate Williams, puisque « Russell, Graetz et Witaenem (...) élaboraient la première théorie du *Game Design* (...) [la] Théorie des Jouets d'Ordinateur » (Williams 2017, 38 TDR)[22]. C'est pourquoi Williams a aussi explicitement listé les trois critères de cette nouvelle théorie :

- « Il doit faire la démonstration, dans la mesure du possible, des capacités matérielles de l'ordinateur, et exploiter celles-ci dans leurs limites ;
- À l'intérieur d'un *Framework* cohérent, il doit rester intéressant [à jouer], ce qui signifie que chaque exécution [du jeu] doit être différente ;
- Il doit faire du spectateur un participant » (*ibid*. TDR)[23].

Pourtant, aucun prototype vidéoludique n'a été commercialisé jusqu'alors. De surcroît, le développement de *Spacewar!* (Russell 1962, PDP-1) n'était pas censé, selon Barton, « créer une nouvelle forme de divertissement (...) [mais] seulement (...) une voie pour montrer (...) [des] compétences en programmation informatique » (Barton 2019, 5 TDR)[24]. Toutefois, il a fallu patienter presque une décennie avant la publication du premier jeu d'arcade commercial, notamment lorsque, comme le rappelle Michael Ray, « Nolan Bushnell et Ted Dabney ont simplifié le jeu (...), et cette version a été publiée (...) sous le nom de *Computer Space* » (Ray 2011, 37 TDR)[25]. Ce dernier jeu est *de facto* considéré comme étant,

21. « two-player spaceship shooting game » (Bossom et Dunning 2015, 20).
22. « Russell, Graetz, and Witaenem (...) formulated the first articulated theory of video game design (...) [the] Theory of Computer Toys » (Williams 2017, 38).
23. 1) It should demonstrate as many of the computer's resources as possible, and tax those resources to the limit 2) Within a consistent framework, it should be interesting [to play], which means that every [game] run should be different 3) It should make the viewer a participant (Williams 2017, 38).
24. « to create a new form of entertainment (...) [but] just (...) a way to show off (...) programming skills » (Barton 2019, 5).
25. « Nolan Bushnell and Ted Dabney simplified the game (...), and this version was published (...) as *Computer Space* » (Ray 2011, 37).

d'après Jean-François Morisse, « l'un des plus anciens jeux vidéo connus à avoir été commercialisé » (Morisse 2013, 14).

Puis, dès le début des années 1970, ce fut la société *Magnavox* qui joua un rôle majeur grâce à un nouveau projet de l'ingénieur Baer, qui cette fois-ci a abouti à l'invention d'une véritable console de jeu. En effet, après le rejet de son concept initial de jeu sur téléviseur en 1950 par *Loral Electronics Corp*, il a, selon Brett Weiss, conçu « en 1967 (...) sa légendaire Brown Box » (Weiss 2007, 247 TDR)[26] pour *Magnavox*. Un même prototype mis sur le marché « en 1972 (...) [sous le nom de] la Magnavox Odyssey (...) la première console au monde à être commercialisée » (*ibid.* TDR)[27].

Cependant, la *Magnavox Odyssey* se révèle encore prototypique en comparaison des consoles d'aujourd'hui. Cela principalement à cause, selon Rusel DeMaria, de son architecture « extrêmement simple (...) [qui] n'utilisait pas des circuits intégrés, mais était plutôt composée de 40 transistors et 40 diodes » (DeMaria 2019b, 18 TDR)[28]. Toutefois, cette console favorisa l'innovation sur le marché des jeux vidéo d'arcade, comme indiqué par Ichbiah, puisque « Bushnell, Alcorn et Dabney ont l'idée d'un jeu qui s'inspirerait de (...) la console Odyssey » (Ichbiah 2014). Nommé *Pong* (Atari 1972, Arcade), ce nouveau jeu a incontestablement contribué à la démocratisation de l'industrie vidéoludique, devenant, pour reprendre les propos de Bossom et Dunning, « un succès immédiat et indubitablement un événement majeur dans l'histoire du divertissement » (Bossom et Dunning 2015, 22 TDR)[29].

Mais la croissance économique de cette industrie balbutiante a néanmoins subi, selon l'étude de Mirko Ernkvist, un premier « tassement [du marché] de l'arcade de 1975 » (Ernkvist 2008, 162 TDR)[30]. Cet événement fut principalement causé, toujours d'après les analyses d'Ernkvist, par une surabondance de jeux vidéo d'arcade profondément similaires. Il explique surtout un phénomène

26. « by 1967 (...) his legendary Brown Box » (Weiss 2007, 247).
27. « in 1972 (...) [under the name of] the Magnavox Odyssey (...) the world's first commercially available video game console » (Weiss 2007, 247).
28. « extremely simple (...) [and] did not use integrated circuits, but rather consisted of 40 transistors and 40 diodes » (DeMaria 2019b, 18).
29. « an instant success and an unmistakable landmark in entertainment history » (Bossom et Dunning 2015, 22).
30. « arcade [market] shake-out of 1975 » (Ernkvist 2008, 162).

particulier, celui dans lequel « toutes les entreprises qui s'introduisaient sur le marché fabriquaient des imitations plus ou moins similaires du Pong originel » (*ibid.*, 9 TDR)[31]. Trop récent, l'industrie ne semblait aucunement établir des normes de qualité ou de créativité pour les différents jeux. Et c'est justement, comme souligné par Ernkvist, ce « manque de capacité d'innovation des entreprises sur le marché combiné avec des contraintes technologiques » (*ibid.* TDR)[32] qui provoquait un désintérêt momentané des joueurs. Cela explique pourquoi, pour reprendre les conclusions d'Ernkvist « l'intérêt des consommateurs pour de nouvelles expériences de divertissement ne pouvait pas être maintenu » (*ibid.*, 10 TDR)[33] suffisamment longtemps.

Malgré cet essoufflement, une pléthore de nouvelles œuvres iconiques remplissait progressivement les salles d'arcade vers la fin des années 1970, une ère communément appelée, comme le rappelle Kent, « l'âge d'or des jeux vidéo » (Kent 2001, 122 TDR)[34]. Mais c'était surtout l'introduction *in illo tempore* du jeu « *Space Invaders* [qui] déclencha l'intérêt des consommateurs aussi bien envers les jeux vidéo que les jeux d'arcade » (*ibid.*, 189 TDR)[35]. Le succès de ce jeu était tellement important au Japon qu'il a potentiellement causé, comme souligné par Mark J. P. Wolf, « une pénurie de pièces de 100 yens, obligeant le gouvernement à tripler leur production » (Wolf 2008a, 41 TDR)[36]. Une information qui semble pourtant être, selon Charles Paradis, « l'un des mythes les plus erronés du début de l'histoire des jeux vidéo » (Paradis 2014, 46 TDR)[37]. Au-delà de cette hypothétique anecdote, l'âge d'or était une période si propice pour l'industrie vidéoludique, selon Ernkvist, qu'en « 1980, le marché [des jeux] d'arcade avait grandement dépassé celui de l'industrie du flipper » (Ernkvist 2008, 182 TDR)[38].

31. « all companies that entered the market made more or less similar imitations of the original Pong » (Ernkvist 2008, 171).
32. « lack of innovation capabilities among the firms in the market combined with technological restraints » (Ernkvist 2008, 171).
33. « consumers' appetite for new entertainment experiences could not be quelled » (Ernkvist 2008, 172).
34. « the golden age of video games » (Kent 2001, 122).
35. « *Space Invaders* [that] ignited interest in consumer video games, as well as arcade games » (Kent 2001, 189).
36. « a shortage in 100-yen coins, causing the government to triple its production of them » (Wolf 2008a, 41).
37. « one of the most erroneous tales of early video game history » (Paradis 2014, 46).
38. « 1980, the arcade [gaming] market had grown bigger than the pinball industry » (Ernkvist 2008, 182).

Nous pouvons ainsi énumérer plusieurs œuvres importantes à l'époque et citées par Kent (2001), notamment *Asteroids* (Atari 1979, Arcade), *Galaxian* (Midway Games 1980, Arcade) ou encore *Pac-Man* (Midway Games 1980, Arcade).

Pendant cette croissance exponentielle, on verra s'implanter dans l'industrie vidéoludique, et selon Kent, « quelques entreprises japonaises [qui] ont réalisé des bénéfices considérables (...) [notamment] Namco (...) grâce à son partenariat avec Midway [ou] Taito » (Kent 2001, 155 TDR)[39]. Mais Nintendo, qui représente aujourd'hui une multinationale majeure de l'industrie vidéoludique, semblait pourtant « incapable de conquérir le marché américain » (*ibid*. TDR)[40]. Or, cette entreprise, « créée en 1889 par Fusajiro Yamauchi pour fabriquer et distribuer des cartes à jouer japonaises dénommées *Hanafuda* » (Novak 2012, 13 TDR)[41], était déjà bel et bien présente dans ce même secteur. C'est notamment le cas grâce à la distribution de sa première console de salon, comme indiqué par Harris, dès « 1977, (...) la Color TV-Game 6, qui jouait six versions légèrement différentes de tennis électronique » (Harris 2015 TDR)[42].

Toutefois, pour pallier ce problème d'absence de succès aux États-Unis, Nintendo s'est attelé à développer un nouveau jeu vidéo, comme indiqué par Bross et Lassinat-Foubert, notamment avec « un jeune illustrateur, Shigeru Miyamoto (...) [et] Gunpei Yokoi (...) [qui] conçoivent alors une adaptation de Popeye » (Bross et Lassinat-Foubert 2018). Cependant, la société japonaise fut dans l'obligation de procéder à une modification du jeu, puisque « la société détentrice des droits [de Popeye] (...) refuse le jeu en l'état » (*ibid*.). C'est ainsi que celui-ci a été renommé *Donkey Kong*, introduisant plusieurs protagonistes désormais célèbres, dont un certain *Mario.* Un même personnage qui représente, aujourd'hui encore, toujours pour reprendre le constat de Bross et Lassinat-Foubert, « la seule mascotte (...) parfaitement identifiable dans le monde du jeu vidéo » (*ibid*.)**.** Cette étape charnière permet surtout de renforcer la présence de Nintendo dans l'industrie vidéoludique. Car, dès la commercialisation du jeu, et

39. « a few Japanese companies [which] earned huge profits (...) [in particular) Namco (...) through its partnership with Midway [or] Taito » (Kent 2001, 155).
40. « unable to crack the American market » (Ken 2001, 155).
41. « established in 1889 by Fusajiro Yamauchi to manufacture and distribute Japanese playing cards known as Hanafuda » (Novak 2012, 13).
42. « 1977, (...) the Color TV-Game 6, which played six slightly different versions of electronic tennis » (Harris 2015).

selon l'analyse de Kent, « en 1981, l'année la plus importante de l'arcade, Donkey Kong était deuxième en termes de popularité après Pac-Man » (Kent 2001, 305 TDR)[43]. Le succès était donc considérable, et près d'une année après sa mise en circulation, on constate, toujours grâce à Kent, que « Nintendo a gagné plus de 180 millions de dollars grâce aux ventes d'environ 60 000 machines d'arcade *Donkey Kong* » (*ibid.*, 215 TDR)[44].

Parallèlement, le marché des consoles de salon poursuivait son expansion puisque, comme l'a indiqué Kent, « en 1975, Atari a commercialisé une version [console] de Pong et est devenu la première entreprise à fabriquer à la fois des produits d'arcade et [de console] » (Kent 2001, 80 TDR)[45]. Pour autant, un phénomène provoqua *de facto* une première saturation sur le marché des consoles, relativement similaire au tassement du marché de l'arcade évoqué plus haut. Ainsi, nous pouvons reprendre les observations d'Amos, indiquant que « la simplicité de la puce de Pong sur un circuit intégré a ouvert les vannes aux fabricants électroniques pour produire leurs propres consoles Pong » (Amos 2021, 21 TDR)[46]. Ernkvist quantifie précisément ce phénomène, où « le nombre de systèmes introduits incluant Pong (...) a augmenté de 1 (Atari) en 1975 à (...) 162 systèmes en 1977 » (Ernkvist 2008, 175 TDR)[47]. Paradoxalement, on observait, grâce à Barton, que la « demande pour Pong était en déclin » (Barton 2019, 18 TDR)[48] sur ce même marché des consoles. Une situation où l'offre semblait donc largement supérieure à la demande, phénomène qui entraîna inévitablement le premier « krach de l'industrie [vidéoludique] à grande échelle en 1977 » (*ibid*. TDR)[49] sur le marché des consoles de salon.

Toutefois, une nouvelle ère vidéoludique a permis de remédier à cette situation de crise économique : la deuxième génération de consoles. Remplaçant

43. « in 1981, the biggest year of arcades, Donkey Kong was second only to Pac-Man in popularity » (Kent 2001, 305).
44. « Nintendo had grossed more than $180 million from sales of approximately 60, 000 *Donkey Kong arcade* machines » (Kent 2001, 215).
45. « in 1975, Atari released a (...) [home] version of Pong and became the first company to make both arcade and (...) [home] products » (Kent 2001, 80).
46. « the simplicity of the Pong on a chip integrated circuit opened the floodgates for electronics manufacturers to produce their own Pong consoles » (Amos 2021, 21).
47. « the number of systems introduced using Pong (...) increased from 1 (Atari) in 1975 to (...) 162 systems in 1977 » (Ernkvist 2008, 175).
48. « demand for Pong was on the decline » (Barton 2019, 18).
49. « full scale [gaming] industry crash in 1977 » (Ernkvist 2008, 175).

rapidement la première génération grâce à des innovations majeures, on constate avec Williams que cette dernière « s'étend approximativement de 1976 à 1983, [où] ces consoles (...) introduisent des concepts de jeu et des pratiques commerciales qui ont façonné (...) l'industrie pendant des décennies » (Williams 2017, 91 TDR)[50]. L'une de ces innovations était notamment apportée, d'après les propos de Kent, par la console « Fairchild (...) *Channel F* en août 1976 (...) [qui] permettait de jouer à des jeux stockés sur des cartouches interchangeables » (Kent 2001, 90 TDR)[51]. Tout en éludant les détails techniques dans cette introduction, spécifions qu'il s'agissait de la première fois qu'une console permettait l'acquisition de nouveaux jeux, au-delà de ceux intégrés dans la machine elle-même.

Peu après, et comme le souligne Williams, la multinationale « Atari a distribué la console à cartouches en 1977 appelée la "Video Computer System" ou VCS (plus tard renommée 2600 (...)) » (Williams 2017, 92 TDR)[52]. Une console élaborée dans la continuité de la Fairchild *Channel F* puisqu'elle « était plus qu'une machine de jeux, c'était un ordinateur » (Kent 2001, 100 TDR)[53]. Grâce à ses capacités techniques, celle-ci, selon Jeannie Novak, « lance avec succès l'industrie des consoles de jeux de salon » (Novak 2012, 14 TDR)[54], et comprenait des œuvres vidéoludiques comme *Space Invaders* (Atari 1980, Atari 2600), *Asteroids* (Atari 1981, Atari 2600) ou encore *Pitfall!* (Activision 1982, Atari 2600).

Malgré une situation éminemment favorable, selon Ichbiah, « en 1982, [où] le marché des jeux représente un volume financier considérable au point d'éclipser les secteurs de loisirs traditionnels » (Ichbiah 2014), l'industrie vidéoludique a ensuite connu sa plus grande crise en 1983. Cette même crise s'explique par une pluralité de facteurs, mais, d'après les frères Frédéric et Alain Le Diberder, elle « était surtout une crise de surproduction typique (...) [et] la concurrence des

50. « Spanning approximately from 1976 to 1983, [where] these (...) consoles introduced game concepts and business practices that shaped the industry (...) for decades » (Williams 2017, 91).
51. « Fairchild (...) Channel F in August 1976 (...) [which] played games stored on interchangeable cartridge » (Kent 2001, 90).
52. « Atari released a cartridge-based console in 1977 called the "Video Computer System" or VCS (later renamed the 2600 (...)) » (Williams 2017, 92).
53. « officially named the "Video Computer System" (VCS), (...) was more than a game machine; it was a computer » (Kent 2001, 100).
54. « successfully launching the home game console industry » (Novak 2012, 14).

micro-ordinateurs était sérieuse tant en Amérique qu'en Europe » (Le Diberder et Le Diberder 1998, 31). Ichbiah souligne principalement l'absence de régulation du marché, puisque « les ventes sont d'un niveau suffisamment élevé pour faire croire que le public sera prêt à acheter n'importe quoi » (Ichbiah 2014). Kent va également mettre en exergue deux échecs majeurs de jeux sur la console *Atari 2600* : *Pac-Man* (Atari 1982, Atari 2600) et *E.T. The Extra-Terrestrial* (Atari 1982, Atari 2600). En effet, il explique que si « Atari a réussi à vendre des millions de cartouches de *Pac-Man* (...) la majorité des cartouches d'*E.T.* sont restées invendues » (Kent 2001, 239 TDR)[55]. Plus encore, il s'agissait de la qualité des titres qui semblait problématique, puisque, pour reprendre les propos de Brian Wardyga, « *Pac-Man* et *E.T. The Extra-Terrestrial* étaient (...) [parmi] les nombreux jeux de mauvaise qualité » (Wardyga 2019, 73 TDR)[56]. Nous pouvons ainsi synthétiser les deux générations de consoles dans le tableau ci-dessous, en nous appuyant principalement sur les données de Wardyga (2019, 20-75).

	Consoles	Année de distribution	Ventes mondiales approximatives
Première génération	**Magnavox Odyssey**	1972	Inférieur à 350 mille
	Atari Home Pong	1975	Supérieur à 550 mille
	Coleco Telstar	1976	Supérieur à 1 million
Seconde génération	**Fairchild Channel F**	1976	Inférieur à 350 mille
	Atari 2600	1976	30 millions
	Magnavox Odyssey²	1978	2 millions
	Intellivision	1982	3 millions
	ColecoVision	1982	Supérieur à 2 millions
	Atari 5200	1982	1 million

Tableau 3. Ventes mondiales de la première et deuxième génération de consoles

55. « Atari had managed to sell millions of *Pac-Man* cartridges (...) the majority of *E.T.* cartridges remained in dead inventory » (Kent 2001, 239).
56. « *Pac-Man* and *E.T. The Extra-Terrestrial* were (...) [among] countless poorly made games » (Wardyga 2019, 73).

De cet événement résulta une restructuration de l'industrie vidéoludique, au cours de laquelle une importance prépondérante fut attribuée à la qualité des jeux. Pour autant, comme le souligne Kent, si « le marché Américain des jeux vidéo semble avoir subi un krach en 1983, (...) le marché international continuait presque sans entrave » (Kent 2001, 278 TDR)[57]. Pendant cette même période, Chris Crawford publia le premier ouvrage théorique sur les jeux vidéo, ayant hypothétiquement pour objectif de structurer l'espace vidéoludique et de prémunir l'industrie de crises similaires à celles de 1975, 1977 et surtout de 1983. De surcroît, il a mis en exergue une typologie majeure, qui posa de nouvelles fondations pour l'interactivité et le *gameplay* : les genres vidéoludiques.

57. « the American video game market may have crashed in 1983, but the international market continued almost unimpeded » (Kent 2001, 278).

1-1-A. TAXINOMIE DES GENRES VIDÉOLUDIQUES

La taxinomie des genres ou types vidéoludiques est constituée d'une nomenclature de descripteurs compendieux, chacun regroupant un ensemble de mécanismes interactifs. Ceux-ci sont principalement, d'après les observations de Svitlana et Alina Bezchotnikova, « basés sur des investigations scientifiques individuelles et tentatives réalisées par des experts de l'industrie vidéoludique » (Bezchotnikova et Bezchotnikova 2018, 1 TDR)[58]. Ils permettent surtout de pouvoir, d'après Ernest Adams, subdiviser le *gameplay* des « jeux (...) dans des ensembles ordonnés de genres que tout le monde reconnaissait » (Adams 2009 TDR)[59]. Ces multiples catégories sont désignées linguistiquement par des sigles tels que *FPS* pour *First-Person Shooter* et RPG pour *Role-Playing Game*, des termes comme *aventure* ou *combat*, ou encore des expressions figées, notamment *beat 'em up* ou *hack'n slash*.

Toutefois, on peut constater qu'il semble être particulièrement difficile de déterminer l'origine précise des premières taxinomies des genres vidéoludiques. Ainsi, on pourrait seulement formuler l'hypothèse que la première occurrence d'un ouvrage théorique sur le jeu vidéo fut rédigée, d'après Wolf et Bernard Perron, « en 1982 [par] Chris Crawford [qui] a écrit *The Art of Computer Game Design*, le premier livre consacré à la théorisation des jeux vidéo (...) publié (...) en 1984 » (Wolf et Perron 2003, 4 TDR)[60]. Comme expliqué par Crawford, le rôle de cette classification était calqué sur le modèle darwinien de l'évolution biologique des espèces puisque, à l'instar du « travail taxinomique méticuleux de Darwin (...), une taxinomie des jeux d'ordinateur doit éclairer sur les facteurs communs qui lient les différentes familles de jeux » (Crawford 1984, 19 TDR)[61]. Il a également précisé que cette même nomenclature distinguait deux catégories majeures, à savoir les « jeux d'adresse et d'action (A&A) (qui insistent sur les

58. « based on individual scientific investigations and attempts made by game industry experts » (Bezchotnikova et Bezchotnikova 2018, 1).
59. « games (...) into a neat set of genres that everybody recognized » (Adams 2009).
60. « in 1982 [by] Chris Crawford [who] wrote The Art of Computer Game Design, the first book devoted to theorizing about video game (...) published (...) in 1984 » (Wolf et Perron 2003, 4).
61. « Darwin's meticulous taxonomic work (...) a taxonomy of computer games should illuminate the common factors that link families of games » (Crawford 1984, 19).

habiletés perceptivo-motrice) et les jeux de stratégie ou cognitifs (qui insistent sur les efforts intellectuels) » (*ibid*. TDR)[62].

Pourtant, Järvinen va mettre en exergue une hypothèse où, malgré la création de cette typologie spécifique par Crawford, « les conceptions populaires de genres proviennent principalement du journalisme de jeux vidéo (...) [et] la taxinomie de (...) Crawford (...) n'a pas trouvé son chemin dans le discours général » (Järvinen 2002 TDR)[63]. Mais, au-delà des interrogations sur son origine, cette catégorisation nouvelle des jeux vidéo attribuait *de facto* une réelle structuration de l'espace vidéoludique, puisque, selon Adams, les « éditeurs ont élaboré des plans produits basés sur eux (...) [et] les joueurs ont appris à préférer un genre plutôt qu'un autre » (Adams 2009 TDR)[64].

Pour autant, le marché du jeu vidéo en 1983 était encore partagé entre une crise localisée aux États-Unis, et une situation plus propice au Japon. C'est, en effet, le constat des frères Le Diberder, qui expliquent qu'une telle situation était induite par le « protectionnisme de fer [qui] empêchait la concurrence étrangère, et les ordinateurs familiaux étaient handicapés par le problème du clavier, inapte à l'écriture japonaise » (Le Diberder et Le Diberder 1998, 31). Ainsi, le marché des consoles de salon a bel et bien subsisté au Japon, avec la commercialisation de deux consoles le même jour, comme le rappelle Wardyga, marquant le début de la troisième génération de consoles : « la Nintendo Famicom (...) [et] la Sega SG-1000 (...), le 15 juillet 1983 » (Wardyga 2019, 116 TDR)[65]. La *Famicom* était perçue comme étant une vraisemblable « preuve d'innovation et d'économie technique » (Kent 2001, 278)[66]. De surcroît, son hypothétique puissance technologique permettait, d'après Kent, de « produire plus de couleurs et des graphismes plus détaillés que (...) n'importe quel système avant elle » (*ibid.*

62. « skill-and-action (S&A) games (emphasizing perceptual and motor skills) and strategy or cognitive games (emphasizing cognitive effort) » (Crawford 1984, 19).
63. « the popular genre conceptions originate mostly from game journalism (...) [and] Crawford (...) taxonomy (...) has not found its way into general discourse » (Järvinen 2002).
64. « publishers created product plans based on them (...) [and] gamers learned to prefer one genre over another » (Adams 2009).
65. « the Nintendo Famicom (...) [and] the Sega SG-1000 (...) on July 15, 1983 » (Wardyga 2019, 60).
66. « a testament to innovation and economic engineering » (Kent 2001, 278).

TDR)[67]. En outre, et comme observé par Ichbiah, cette console présentait une caractéristique favorisant une plus grande popularité : un « prix deux fois inférieur à celui des machines concurrentes » (Ichbiah 2014). Kent a constaté qu'elle a connu, grâce à cette situation, « un succès spectaculaire (...) [avec] plus de 500 mille Famicom [vendues] en deux mois » (Kent 2001, 279-280 TDR)[68].

Désormais, grâce à « son succès avec les consommateurs japonais (...) [Nintendo] ambitionne d'exporter sa console de jeux aux États-Unis » (*ibid.*, 280 TDR)[69], mais avec plus de prudence qu'à l'accoutumé. En effet, ce territoire était encore, d'après les constatations de Levy et Novak, un « marché traumatisé par des mauvais jeux » (Levy et Novak 2010, 11 TDR)[70], une situation découlant du krach vidéoludique de 1983. Pour Nintendo, cela impliquait *de facto* l'élaboration d'une nouvelle stratégie de commercialisation de la console *Famicom,* en modifiant de manière substantielle plusieurs de ses aspects. Grâce à Wardyga, on peut particulièrement se focaliser sur l'appellation de celle-ci, où Nintendo a « renommé la console (...) **Nintendo Entertainment System** (**NES**) » (Wardyga 2019, 111 TDR)[71], tout autant que sur la métamorphose visuelle de la console. Car, loin d'arborer une esthétique classique d'une console de jeu, et d'après Damien McFerran, Nintendo ajouta à la « NES (...) un système de chargement frontal de cartouches qui ressemblait à un magnétoscope » (McFerran 2019 TDR)[72], occultant la fonction première de la console : le jeu vidéo ; du moins, visuellement seulement. Ces stratagèmes permettaient, d'après Ichbiah, de se différencier des consoles précédentes, avec un « message (...) limpide : Nintendo promet des jeux (...) de qualité (...) [et] une véritable plateforme de divertissement » (Ichbiah 2014). L'objectif était *in fine* de prémunir l'industrie d'une crise similaire à celle de 1983 et, par la même occasion, de rasséréner les clients potentiels sur cette machine.

67. « produce more colors and more detailed graphics than (...) any system before it » (Kent 2001, 278).
68. « a spectacular success (...) [with] more than 500,000 Famicom [sold] within two months » (Kent 2001, 279-280).
69. « his success with Japanese consumers, (...) [Nintendo] set his sights on exporting his game console to the United States » (Kent 2001, 280).
70. « market traumatized by bad games » (Levy et Novak 2010, 11).
71. « renamed the console (...) Nintendo Entertainment System (NES) » (Wardyga 2019, 111).
72. « NES (...) a front-loading cartridge system which resembled a VCR » (McFerran 2019).

Néanmoins, la clef de voûte de cette stratégie globale reposait surtout sur un autre aspect, également souligné par McFerran, le « '*Original Nintendo Seal of Quality*', un badge blanc et doré (...) pour assurer aux consommateurs qu'ils avaient un produit d'excellente qualité » (McFerran 2019 TDR)[73]. Pouvant être toutefois perçu, à tort, comme un symbole de qualité pour un jeu, Chris Scullion va désambiguïser le rôle de ce label. En effet, il indique que le « *Seal of Quality* (...) était simplement une promesse que le jeu répondait aux normes de contrôle de qualité de Nintendo ; (...) un niveau basique de compétences en programmation (...) et était simplement fonctionnel » (Scullion 2019, 12 TDR)[74]. Il était surtout question de garantir *in fine* le bon fonctionnement technique d'une œuvre, plutôt que la qualité globale du produit pour les consommateurs. Toutefois, on pourrait hypothétiquement penser que cette ambivalence du *Seal of Quality* pour les joueurs, autrement dit une interprétation basée sur la qualité du produit plutôt que sur son simple fonctionnement, était avantageuse pour Nintendo. En substance, l'émergence de nouvelles règles et normes de qualité des jeux vidéo provoqua une véritable métamorphose de l'industrie vidéoludique selon Levy et Novak, puisque c'était « la première fois qu'un fabricant imposait de manière stricte les lignes directrices du contenu » (Levy et Novak 2010, 11 TDR)[75].

Dans tous les cas, ces standards de qualité ont impliqué une importante restructuration du marché, notamment avec une taxinomie des genres vidéoludiques qui s'était *de facto* imposée pour la *NES*. Un phénomène que Doucet peut illustrer via « des marqueurs formels du genre dans deux endroits : les magazines "*Players Guide*" (...) et la série "*Black Box*" des premiers titres NES » (Doucet 2020 TDR)[76]. Doucet liste également ces « 30 jeux (...) publiés sous ce format de boîte de jeu » (*ibid*. TDR)[77] dans un tableau que nous

73. « 'Original Nintendo Seal of Quality', a white-and-gold badge (...) to assure consumers that they were getting a top-notch product » (McFerran 2019).
74. « Seal of Quality (...) was simply a promise that the game had met Nintendo's quality control standards ; (...) a basic level of programming competence (...) and essentially just worked » (Scullion 2019, 12).
75. « the first time a manufacturer enforced strict content guidelines » (Levy et Novak 2010, 11).
76. « formal genre markers in two places : "Players Guide" magazines (...) and the "Black Box" series of early NES titles » (Doucet 2020).
77. « 30 games (...) released under this packaging format » (Doucet 2020).

reproduisons ci-dessous et qui permet de mettre en surbrillance les multiples genres de l'époque.

Genre	Jeux vidéo
Action	Balloon Fight, Clu Clu Land, Ice Climber, Kung Fu, Pinball, Super Mario. Bros, Urban Champion
Adventure	Zelda (1 & 2), Metroid, Castlevania, Kid Icarus
Arcade	Donkey Kong (1-3), Mario Bros., Popeye
Sports	10-Yard Fight, Baseball, Golf, Pro Wrestling, Slalom, Soccer, Tennis, Volleyball
Education	Donkey Kong Jr. Math
Light Gun	Duck Hunt, Gumshoe, Hogan's Alley, Wild Gunman
Programmable	Excitebike, Mach Rider, Wrecking Crew
Robot	Gyromite, Stack-up

Tableau 4. Genres des 30 jeux NES de la série *Black Box* (Doucet 2020)

Dès lors, Christian Elverdam et Espen Aarseth constatent que l'industrie s'est retrouvée dans une « situation où les jeux sont classés par des genres arbitraires, contradictoires ou qui se chevauchent » (Elverdam et Aarseth 2007, 4 TDR)[78]. Des classifications qui pouvaient déjà susciter de multiples interrogations, notamment lorsqu'on avait la possibilité de classer un jeu aussi bien dans la catégorie *arcade* ou *sport* que dans le genre *action*. Surtout, Bossom et Dunning relèvent que les variations catégorielles au sein des genres vidéoludiques semblent être liées à la nature intrinsèque de cette taxinomie, puisque « la plupart du temps, les genres de jeux ne sont pas purs, mais plutôt des hybrides avec de nombreuses caractéristiques qui se chevauchent » (Bossom et Dunning 2015, 38 TDR)[79]. Plus encore, c'est surtout, constate Dominic Arsenault, « l'imprécision fondamentale du genre (...) [qui] constitue en fait sa spécificité » (Arsenault 2011,

78 . « situation where games are classified by arbitrary, contradictory, or overlapping genres » (Elverdam et Aarseth 2007, 4).

79. « most often game genres are not pure breeds, but are instead hybrids with many overlapping traits » (Bossom et Dunning 2015, 38).

333), offrant paradoxalement une malléabilité naturelle et une adaptabilité pour classer les différents produits.

Cependant, l'interrogation cruciale que nous cherchons à mettre en surbrillance dans notre étude est la présence ou l'influence potentielle de cette même taxinomie dans l'étape de création de jeux vidéo. Plus précisément lors de l'élaboration des différents mécanismes interactifs liés au *Game Design*, c'est-à-dire à ce qui, indique Richard Rouse III, « détermine la forme du *gameplay* (...) quelles décisions (...) quels critères de victoire ou de défaite (...) [en résumé], comment le *gameplay* fonctionnera » (Rouse III 2005, XXI TDR) [80] . Les caractéristiques principales du *Game Design* sont souvent rédigées dans des documents nommés *Game Design Documents* (désormais GDD). Briar Lee Mitchell précise que le GDD est habituellement conçu comme étant un « document *fluide* (...) [contenant] un référentiel de l'ensemble des éléments intégrés dans un jeu, y compris l'art, les dialogues, l'audio, et ainsi de suite » (Mitchell 2012, 183 TDR)[81]. Généralement, on constate grâce à Rouse III que « il n'existe aucun format spécifique » (Rouse III 2005, 356 TDR) [82] ni aucune standardisation pour la rédaction du GDD, mais un tel document implique néanmoins de classer le projet de jeu dans un genre établi. En d'autres termes, il est question de fournir un descriptif laconique du type de *gameplay*, selon Mitchell, et simplement « sous forme de description d'un à trois mots, comme (...) *action-aventure* » (Mitchell 2012, 56 TDR)[83].

Pourtant, Novak spécifie que, lorsque le développement d'un jeu « combine des genres (création d'un hybride) ou en invente un nouveau (...) l'industrie pourrait considérer cela comme un projet risqué » (Novak 2012, 383 TDR)[84]. Un phénomène qui semble imposer un certain conformisme aux genres existants, paradoxalement imprécis. Ils dépassent ainsi le simple rôle de descripteurs, indique Novak, puisque, dans certains cas, « les éditeurs et investisseurs

80. « determines the form of the gameplay (...) what choices (...) what win or loss criteria (...) [in summary] how the gameplay will function » (Rouse III 2005, XXI).
81. « *fluid* document (...) [containing] a repository of all the elements that go into the game, including art, dialogue, audio, and so on » (Mitchell 2012, 183).
82. « there is no format » ((Rouse III 2005, 356).
83. « written out as a one- to three-word description like (...) *action-adventure* » (Mitchell 2012, 56).
84. « combining genres (creating a hybrid) or inventing a new one (...) the industry might consider this is a risky undertaking » (Novak 2012, 383).

financiers doivent être convaincus que (...) un genre hybride innovant ou tout nouveau sera un succès » (*ibid*. TDR)[85]. En résumé, une contradiction apparente semble émerger au sein de la typologie des genres de jeux vidéo : d'une part, ils sont peu définis et incertains, mais d'autre part, ils semblent jouer un rôle véritablement significatif dans le développement, la réception et l'utilisation des jeux vidéo. C'est donc à partir de ce potentiel paradoxe dans le développement de jeux vidéo qu'il est possible d'esquisser une problématique centrale, structurant intégralement cette thèse.

85. « publishers and financial backers might need to be convinced that (...) innovative hybrid or brand new genre will be a success » (Novak 2012, 383).

1-1-B. PROBLÉMATIQUE POLYMORPHIQUE

À l'aune de l'historique vidéoludique tracé jusqu'à présent, depuis les premiers prototypes interactifs des années cinquante jusqu'aux consoles *NES* et *SG-1000* de la troisième génération de consoles, nous constatons une évolution véritablement substantielle du jeu vidéo. Celle-ci implique une multitude d'innovations technologiques majeures, contournant des limitations qui semblaient, à de multiples égards, infrangibles. Surtout, le processus de développement de jeux vidéo s'est indéniablement standardisé après le krach de l'industrie vidéoludique de 1983. Cela surtout grâce à une norme de la qualité technique et fonctionnelle des jeux, élaborée et imposée par Nintendo avec son *Seal of Quality*. Néanmoins, cet historique indubitablement succinct ne saurait mettre en lumière l'avancée qui nous semble être la plus significative depuis la genèse des jeux vidéo : l'émergence des mondes virtuels en trois dimensions. Apparaissant au milieu des années 1990, comme nous le verrons dans le troisième chapitre, elle a entraîné une forte reconfiguration des mécanismes interactifs et *in fine* de la typologie des genres vidéoludiques.

Des genres qui semblaient être en adéquation avec le contexte d'époque des années 1980, lorsque les jeux vidéo et machines étaient relativement limités sur le plan technique. Or, la perdurance des genres pourrait sembler paradoxale pour des jeux vidéo publiés après les années 2015. En effet, la puissance technologique des consoles et ordinateurs est aux antipodes de celle des décennies précédentes, alors même que la classification des genres semble conserver une forme simplificatrice de l'interactivité. Plus encore, cette typologie pourrait potentiellement symboliser un système fermé pour structurer le *gameplay* d'aujourd'hui, avec des descripteurs qui pourraient sembler, du moins en apparence, inadéquats.

De surcroît, ce classement est encore utilisé par plusieurs acteurs majeurs du paysage vidéoludique, depuis les multiples boutiques en ligne de jeux vidéo jusqu'aux publications destinées aux développeurs. Pour illustrer notre propos, nous avons décidé de répertorier ces différentes sources exploitant les genres vidéoludiques en *Annexe C : Présence des genres vidéoludiques*, en listant de manière exhaustive tous les genres mentionnés. Ainsi, nous avons cité les boutiques *Epic Games Store*, *GOG*, *Microsoft Store, Nintendo eShop*, *PlayStation*

Store et également *Steam*. Mais nous avons aussi référencé le rapport annuel du moteur *Unity*, nommé *Unity Gaming Report 2023*. Cela permet de constater l'usage encore important de ces différents genres dans l'espace vidéoludique. Mais, au-delà de ces multiples sources, cette catégorisation est également présente dans le discours journalistique. Chad Tagbar a notamment illustré ce phénomène à travers la description de la saga « Tekken [qui] est l'un des piliers du genre de jeux de combat » (Tagbar 2023 TDR)[86]. Un autre exemple mis en avant par Michael Higham est celui du jeu *Final Fantasy XVI*, annoncé comme étant « un action RPG qui place fortement l'accent sur la partie "action" » (Higham 2023 TDR)[87]. Cette même taxinomie schématique semble donc largement répandue dans l'ensemble du marché vidéoludique, et pourrait potentiellement avoir de véritables conséquences sur l'évolution ou la limitation de l'interactivité.

Cependant, notre objectif dans cette recherche est d'aller bien au-delà des analyses et interprétations discursives des journalistes. En effet, plutôt que de réaliser une étude portant essentiellement sur cette littérature, nous souhaitons d'emblée examiner méticuleusement des outils informatiques utilisés pour la création de jeux. À travers ce procédé, nous pourrons vérifier l'existence (ou non) des genres vidéoludiques au sein de ces mêmes outils, et examiner *de facto* l'influence potentielle des genres sur le processus de création du *gameplay*. De surcroît, la combinaison de multiples mécanismes interactifs dans les univers vidéoludiques en monde ouvert, une tendance évoquée en introduction, semblerait être en opposition face à une catégorisation potentiellement fermée et cloisonnée des genres. Désormais, à l'aune de l'historique des jeux vidéo tracé jusqu'ici et des évolutions de la taxinomie des genres vidéoludiques, on peut esquisser une première version de la problématique majeure de notre thèse doctorale :

Comment la création des mécanismes interactifs dans les jeux vidéo en 3D temps réel est-elle symbolisée, structurée, élaborée et limitée par les genres vidéoludiques ?

Tableau 5. Ébauche de la problématique de recherche

86. « Tekken [which] is one of the pillars of the fighting game genre » (Tagbar 2023).
87. « an action RPG with a heavy emphasis on the "action" part » (Higham 2023).

Loin de représenter un simple bloc monolithique, nous segmenterons au fil de la recherche cette même problématique en plusieurs composantes distinctes, possédant chacune cette structure bicéphale indécomposable, à la fois sémiologique et informatique. En effet, notre étude entière repose sur ces deux disciplines mentionnées en amont, mais que nous définirons et justifierons plus loin dans cette introduction. Et, afin de s'engager dans des pistes concrètes face à cette problématique clairement énoncée, il est nécessaire de circonscrire notre recherche, par rapport à son aspect sémio-informatique. Pour cela, nous pouvons établir une véritable problématique, plus proche des aspects techniques que nous envisageons, et que nous exposons ci-dessous.

À l'aune des moteurs de jeu des années 2015, comment la création des mécanismes interactifs dans le développement de jeux vidéo en 3D temps réel et depuis les années 1990 est-elle symbolisée, structurée, élaborée et limitée par les genres vidéoludiques ?

Tableau 6. Problématique de la recherche doctorale

Au regard de cette problématique aussi complexe, il est important de poser les jalons d'une hypothèse de départ, à l'aune de nos connaissances du développement de jeux vidéo. Pour cela, il est nécessaire d'examiner les approches scientifiques qui permettent de formuler une telle hypothèse. Et, dans une étude scientifique, Miller et Brewer indiquent qu'il est justement d'usage de partager les « recherches (...) empiriques (...) dans le développement ou le perfectionnement d'idées conceptuelles ou de théories (...) en termes de déduction et d'induction » (Miller et Brewer 2003, 1 TDR)[88]. Ainsi, de ces deux modes de raisonnement découlent *de facto* deux approches scientifiques majeures exploitables pour notre hypothèse : empirico-inductive et hypothético-déductive. On considère que l'approche hypothético-déductive implique, toujours pour reprendre Miller et Brewer, des « *généralisations* (...) à partir desquelles les hypothèses sont déduites, qui sont ensuite testées via des observations empiriques (..) pour confirmer ou infirmer des cas » (*ibid.*, 67 TDR)[89]. Or,

88. « empirical (...) research in the development or refinement of conceptual ideas or theories (...) in terms of deduction and induction » (Miller et Brewer 2003, 1).
89. « *generalisations* (...) from which hypotheses are deduced, which are then tested in empirical observations (..) for confirming or falsifying cases » (Miller et Brewer 2003, 68).

l'approche empirico-inductive est bien différente, dans la mesure où « les généralisations empiriques et les énoncés théoriques devraient être dérivés des données » (*ibid.*, 154 TDR)[90]. En substance, l'induction représente une version véritablement « inverse de la déduction » (*ibid.*, 154 TDR)[91]. En revanche, ces modes de raisonnement, ici inductif ou déductif, ainsi que les modèles principaux pour réaliser les recherches, empirico-inductif et hypothético-déductif dans notre cas, semblent surtout être des modèles schématiques ; ils ne reflètent pas l'exhaustivité des possibilités dans une recherche concrète. De plus, il est possible de retrouver d'autres modes d'inférence, par exemple l'abduction, plutôt que l'opposition binaire formulée précédemment.

À ce propos, il est intéressant de noter que Loiseau (2020) va développer une schématisation de l'épistémologie d'une démarche scientifique plus nuancée. Cela à travers les deux modèles principaux cités précédemment, mais en ajoutant les approches quantitatives ou qualitatives d'une recherche. Il va ainsi considérer que la « diversité des méthodes peut être envisagée selon deux axes fondamentaux, ce qui facilite la construction d'une typologie des méthodes communément utilisées » (Loiseau 2020, 8 TDR)[92]. Nous reprenons ci-dessous le schéma en question, que nous estimons pertinent et utile pour éclairer les principes intrinsèques de notre recherche.

90. « empirical generalisations and theoretical statements should be derived from the data » (Miller et Brewer 2003, 154).
91. « inverse of deduction » (Miller et Brewer 2003, 154).
92. « diversity of methods can be envisaged along two fundamental lines, making it easier to construct a typology of the methods commonly used » (Loiseau 2020, 8).

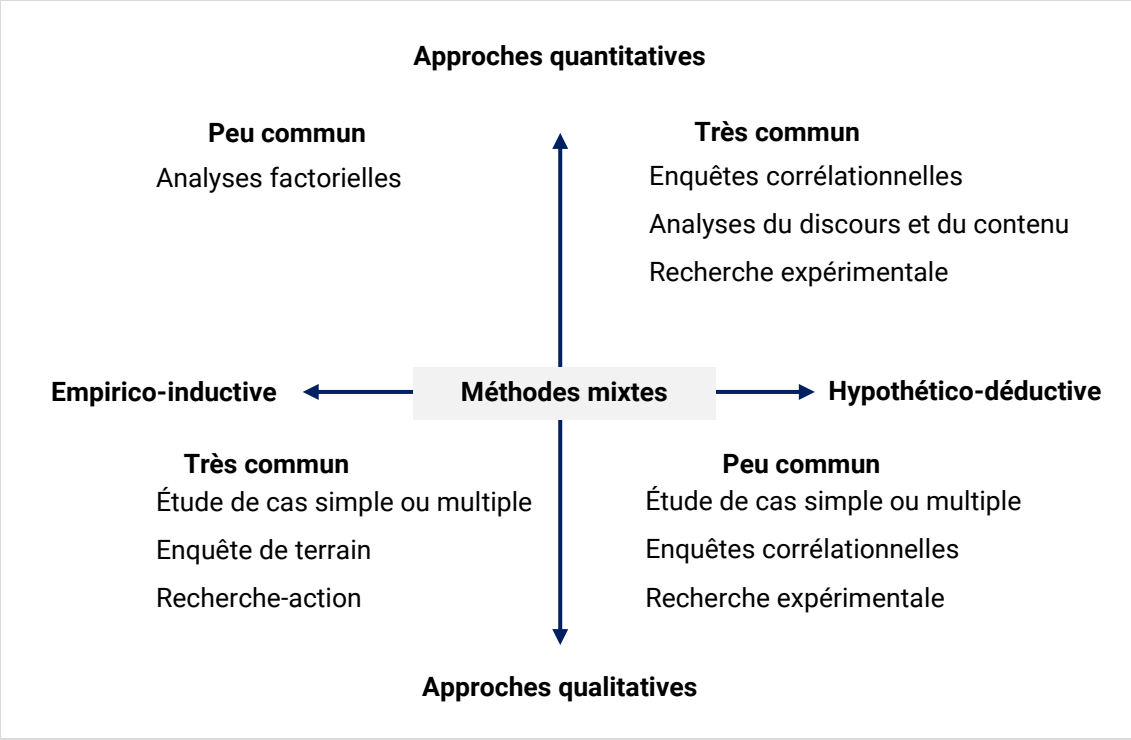


Figure 1. Axes fondamentaux de la démarche scientifique (Loiseau 2020, 9)[93]

Cette perspective bidimensionnelle permet donc de nuancer les oppositions primaires, ajoute Miller et Brewer, puisque « le contraste entre la déduction et l'induction n'est pas aussi marqué que ce qui est souvent suggéré » (Miller et Brewer 2003, 67 TDR)[94]. Plus encore, Miller et Brewer vont plus loin dans ce constat, en indiquant que « les processus d'inférence logique dans la recherche réelle sont, en réalité, plutôt désordonnés » (*ibid.*, 2 TDR)[95]. Ce que nous souhaitons souligner dans cette partie, c'est que ces modèles sont, par essence, fluctuants. Ils peuvent s'adapter à la fois en fonction des recherches et des contextes, s'éloignant *de facto* de modèles mathématiques, par exemple.

Quoi qu'il en soit, nous souhaitons structurer les prémices de cette recherche selon le modèle hypothético-déductif, tout en utilisant une approche qualitative pour la suite de notre démarche scientifique. Autrement dit, nous nous appuyons sur un modèle peu habituel selon le schéma de Loiseau exposé ci-dessus. L'hypothèse de départ qui découle de ce modèle hypothético-déductif sera *in fine* infirmée ou confirmée en conclusion de notre thèse, en fonction des résultats de

93. Schéma original disponible en *Annexe A : Schémas originaux*.
94. « the contrast between deduction and induction is not as stark as it is often suggested » (Miller et Brewer 2003, 67).
95. « processes of logical inference in real research are in fact quite messy » (Miller et Brewer 2003, 2).

notre recherche lors de l'étude sur les corpus. C'est pourquoi nous pouvons désormais clairement énoncer notre hypothèse de départ : nous considérons les genres vidéoludiques comme étant une classification intrinsèquement conventionnelle et arbitraire, c'est-à-dire ne reflétant pas en 2015 des limitations techniques liées aux outils de développement. Une conventionnalité élaborée indépendamment par des développeurs, journalistes et joueurs, avec des définitions incertaines et nuances presque subjectives de chaque genre ; une nomenclature qui fluctue également à travers le temps et l'espace, où un genre en 1990 ne possède pas la même acception que dans les années 2015, par exemple. Surtout, les genres sont définis, altérés et redéfinis à travers les différents jeux vidéo commercialisés.

Par ailleurs, on constate une augmentation du nombre de mécanismes interactifs dans les jeux en monde ouvert, et cela de manière indépendante de la classification des genres. Un phénomène qui semble être, du moins en apparence, un argument supplémentaire en faveur de leur hypothétique conventionnalité. Et, grâce à nos connaissances techniques sur les outils de développement, nous considérons que les genres n'ont aucune véritable existence dans des moteurs de jeu comme *Unreal Engine*. Certes, il existe des mentions résiduelles du genre FPS dans ce même moteur, que nous présenterons et contextualiserons. Mais nous estimons qu'elles sont liées à des raisons historiques plutôt que limitatives et qu'elles n'influencent aucunement la création des mécanismes interactifs. Nous pouvons ainsi synthétiser cette hypothèse de la manière suivante :

Les genres vidéoludiques reflètent des conventions entièrement arbitraires provenant de développeurs, joueurs, jeux vidéo commercialisés et journalistes. Ces conventions ne résultent aucunement de limitations technologiques du développement de jeux vidéo en 3D temps réel dans les années 2015.

Tableau 7. Hypothèse de départ de la recherche doctorale

Au-delà de cette hypothèse, nous pouvons également mettre en exergue l'aspect fondamentalement diachronique de la problématique exposée en amont. En effet, cette recherche porte sur le développement de jeux depuis les années 1990 jusqu'à la date de clôture de cette thèse, autrement dit en 2024. Or, à l'instar des deux modèles du CLG, et pour reprendre les propos de Sandrine

Zufferey et Jacques Moeschler, « l'opposition entre la linguistique synchronique et diachronique » (Zufferey et Moeschler 2021, ch. 5), nous pouvons envisager deux études potentielles du développement des jeux vidéo en 3D : synchronique et diachronique. Mais, contrairement au CLG qui conçoit une « étude de la langue (...) synchronique plutôt que diachronique (*ibid.*), nous estimons qu'il est bénéfique d'utiliser en symbiose ces deux aspects pour étudier les outils de création de jeux vidéo en 3D temps réel. Et cette synergie est particulièrement adaptée à notre problématique qui possède *de facto* un aspect diachronique et synchronique. C'est ainsi que nous pouvons fragmenter l'analyse en deux parties, une première consacrée au « passage d'un état (...) à un autre (...) [dans] une relation de **successivité** » (*ibid.*), exigeant ainsi des comparatifs nombreux entre les multiples outils de développement de logiciels. Puis nous étudierons un outil de développement contemporain en synchronie. Néanmoins, l'aspect diachronique soulève une interrogation fondamentale : quelle périodisation cohérente avec l'évolution des œuvres vidéoludiques est-il possible d'utiliser ? Les premières pistes d'une réponse à cette question se trouvent dans la segmentation opérée par Wolf, qui va distinguer au moins cinq périodisations vidéoludiques différentes :

- « Selon des délimitations arbitraires (décennies, ères, etc.) ;
- Selon une technologie (génération de consoles de salon, machines d'arcade, ou selon les technologies informatiques) ;
- Selon l'industrie [vidéoludique] (sociétés, tendances, krachs, etc.) ;
- Selon des innovations (avancées majeures, éléments pionniers, jeux influents et consoles influentes) ;
- Selon des dominantes (les facteurs influents à un moment donné) » (Wolf 2012b, 2 TDR)[96].

Au regard de notre problématique qui porte sur l'évolution des mécanismes interactifs et outils de développement de jeux vidéo 3D, c'est-à-dire sur des critères éminemment *technologiques*, la segmentation de l'analyse en fonction

96. « • by arbitrary means (decades, eras, etc.) • by technology (generations of home systems, arcade systems, or computer technology) • by [video game] industry (companies, trends, crashes, etc.) • by innovation (breakthroughs, firsts, influential games and game systems, etc.) • by dominants (whichever factor is most influential at a given point in time) » (Wolf 2012b, 2).

de la *technologie* nous paraît être la plus cohérente parmi celles énumérées. En clair, l'objectif est de mettre en surbrillance différents paliers technologiques limpides et représentatifs de l'évolution, aussi bien pour des néophytes que pour des spécialistes. Cela élimine *de facto* une segmentation temporelle par différentes architectures des processeurs, par exemple, notamment à cause d'une évolution bien trop rapide, surtout par rapport à celle des outils de développement de jeux vidéo.

C'est également la raison pour laquelle nous considérons que la segmentation technologique la plus pertinente, du moins au regard du contexte de cette recherche, est celle des générations de consoles. Une périodisation qui nous paraît simple et limpide, se caractérisant par une représentation de l'évolution technologique sous forme de paliers, symbolisant des évolutions technologiques substantielles. Cette délimitation temporelle est, dans certains cas, au cœur même de la conception des consoles. C'est ce qu'il est possible de constater selon les propos de Marc Cerny, le « créateur polyvalent (...) [et] l'architecte principal de la PlayStation 5 » (Ivan 2020 TDR)[97], qui indique être « un grand adepte des générations de consoles » (Cerny 2020, 0:36-0 :38 TDR)[98].

Toutefois, cette segmentation temporelle, à connotation technocentriste, est vectrice de multiples débats remettant parfois en question sa pertinence. D'abord, Dozo va mettre en exergue un hypothétique problème dans cette périodisation, où elle semblerait impliquer « qu'à l'arrivée d'une nouvelle génération, l'ancienne disparaît » (Dozo 2019, 23-24). Autrement dit, elle ne prendrait pas en considération les joueurs qui « n'ont pas nécessairement les moyens de suivre cette succession (...) ou qui préfèrent continuer à s'investir dans un jeu qui n'existe pas sur la nouvelle plateforme » (*ibid.*). Si cette considération n'a aucune pertinence pour notre recherche, puisque nous étudions les outils de développement uniquement, nous nous permettons néanmoins de nuancer quelque peu cet argument. En effet, si les anciennes générations disparaissent bien pour l'industrie vidéoludique, où les développeurs

97. « multi-skilled creator (...) [and] chief architect of PlayStation 5 » (Ivan 2020).
98. « a big believer in Consoles generations » (Cerny 2020, 0:36-0 :38).

doivent concevoir des jeux sur de nouvelles technologies, elles ne remplacent jamais instantanément les générations précédentes.

Nous pourrons illustrer ce phénomène dans notre étude, à travers l'exemple de la console *PlayStation 3* qui, comme nous le verrons par la suite, fut distribuée en 2006. Or, Ryan a constaté que, « en 2009 la PS2 continuait de surpasser les ventes de la PS3 certains mois » (Ryan 2011, ch. 23 TDR)[99]. Naturellement, nous pourrions relever une pluralité de raisons pour expliquer ce phénomène, notamment des raisons financières. Mais, dans tous les cas, nous constatons que la transition vers une nouvelle génération de consoles peut réellement s'étendre sur plusieurs années, ce qui peut sembler considérable à l'échelle de l'évolution technologique. C'est pour cette raison que nous contestons fortement l'idée de Dozo, selon laquelle il serait possible de pouvoir « isoler chaque génération dans une forme d'autarcie bornée » (Dozo, 2019, 24). Bien au contraire, nous considérons *in fine* que les différentes générations peuvent coexister dans un même espace-temps. Cependant, nous ne souhaitons pas approfondir davantage ce phénomène de cohabitation des générations, car celui-ci va au-delà du cadre imposé par notre recherche. En effet, cela impliquerait d'aborder des aspects tels que la situation économique des entreprises de l'industrie vidéoludique ou encore le pouvoir d'achat des joueurs, ce qui nous éloignerait inexorablement de notre problématique.

Ajoutons également que le passage immédiat à une génération ultérieure ne semble pas non plus, d'après Daim *et al.*, concorder avec les intérêts financiers et commerciaux des « fabricants de consoles [qui] voudraient maximiser la durée entre les produits de chaque génération afin de minimiser le nombre de perte de produits et le lancement de matériel à faible marge qu'ils doivent supporter » (Daim *et al.* 2013, 2 TDR)[100]. Plus encore, les développeurs doivent souvent se former à une nouvelle technologie, procédé chronophage et onéreux qui, par essence, est susceptible d'engendrer de nombreux problèmes dans le développement. Dans tous les cas, nous estimons que, même si cette

99. « in 2009 the PS2 was still outselling the PS3 certain months » (Ryan 2011, ch. 23).
100. « a console manufacturer [who] would like to maximize the length of time between product generations in order to minimize the number of loss leaders and low-margin hardware introductions they have to absorb » (Daim *et al.* 2013, 2).

périodisation n'est pas nécessairement objective ni dénuée d'imperfections, elle demeure celle présentant les meilleurs compromis au regard de notre problématique technologique.

Dorénavant, nous pouvons préciser les générations concernées par notre problématique et par notre recherche entière : nous débuterons notre étude dès la cinquième génération de consoles et la terminerons avec la huitième génération. Nous utiliserons néanmoins la quatrième génération en guise d'introduction au début du deuxième chapitre. La justification de l'utilisation d'une délimitation débutant à la cinquième génération découle principalement de l'avènement de la 3D en temps réel, au cœur de notre problématique. Notons également que nous préciserons cela bien plus longuement lors des multiples chapitres concernés, où les chapitres ultérieurs représentent chacun une génération de consoles, hormis le deuxième chapitre qui développe notre approche novatrice. Par ailleurs, nous éludons intégralement la neuvième génération de consoles, car nous estimons qu'il n'est pas possible d'analyser une génération en cours de développement, et dont les perspectives sont encore incertaines. De surcroît, nous préférons éviter les spéculations qui n'ont pas leur place dans notre approche scientifique. Enfin, cette spécificité de segmentation par génération de consoles occasionne la constitution de deux corpus distincts que nous allons présenter dès la prochaine sous-section.

1-1-C. CORPUS DÉVELOPPEMENTAL ET VIDÉOLUDIQUE

Afin d'aborder efficacement la problématique diachronique de cette thèse que nous venons d'exposer, il était impératif de développer notre nouvelle approche scientifique rigoureuse et robuste du Développement Logiciel Tridimensionnel, mentionnée en amont. Celle-ci s'est avérée relativement complexe à élaborer, d'abord en raison des nombreux enjeux qu'elle doit prendre en considération dans le développement de jeux vidéo en 3D temps réel, mais également à cause de l'absence d'une méthodologie déjà existante pour répondre à notre problématique, du moins à notre connaissance. C'est pourquoi elle sera longuement exposée seulement dans le deuxième chapitre, puisque nous considérons qu'elle représente une partie des résultats de nos recherches. Dans la sous-section actuelle, nous allons plutôt présenter les deux corpus distincts qui permettent d'étudier l'évolution complète des mécanismes interactifs : le corpus développemental et le corpus vidéoludique.

Tout d'abord, le corpus développemental est constitué d'outils de développement de jeux vidéo, principalement des moteurs de jeu et langages de programmation. On subdivise d'emblée ce même corpus en deux sous-parties : le corpus développemental direct et le corpus développemental indirect, une distinction inspirée par la classification explicitée par Umberto Eco des sources directes et indirectes. En l'occurrence, Eco indique qu'une « *source directe* est une édition originale ou une édition critique d'un travail » (Eco 2015, 50 TDR)[101], alors que « les œuvres critiques d'autres auteurs, peu importe leur richesse en citations, (...) sont des sources indirectes » (*ibid.* TDR)[102].

Ajoutons également certaines précisions sur les spécificités informatiques ou techniques, que nous incorporons dans ce corpus développemental bicéphale. Ainsi, si nous transposons ces deux concepts dans le domaine de l'informatique, alors nous pouvons d'abord définir le corpus développemental direct comme étant principalement constitué d'un moteur de jeu utilisable pour le grand public, *Unreal Engine 4*, ainsi que ses différents langages de programmation. Ce moteur

101. « *direct source* is an original edition or a critical edition of the work » (Eco 2015, 50).
102. « the critical works of other authors, no matter how rich with quotations, (...) are indirect sources » (Eco 2015, 50).

sera analysé selon une démarche qualitative et en fonction de l'utilisation d'outils similaires dans le milieu professionnel. Nous précisons en amont que le choix se porte sur la quatrième version *d'Unreal Engine* plutôt que la cinquième, notamment par rapport à notre décision de nous restreindre à la huitième génération dans l'analyse diachronique. En effet, nous préférons éluder des analyses portant sur un moteur en cours de développement, autrement dit *Unreal Engine 5*, et de surcroît lié à une génération également en cours d'évolution. Nous estimons qu'il est préférable d'étudier des générations dépassées, afin d'éviter des éventuelles spéculations sur des technologies qui ne sont pas pleinement développées. Toutefois, notons que nous utiliserons dans certains cas la documentation du moteur *Unreal Engine 5*, lorsque les informations concerneront à la fois *Unreal Engine 4* et *Unreal Engine 5*.

Le corpus développemental indirect, quant à lui, est plutôt formé par des outils obsolètes et inaccessibles, qu'il est exclusivement possible d'analyser indirectement, c'est-à-dire à travers des données discursives et via un point de vue purement historique. Ce dernier ne pourra être défini qu'au gré des recherches, et nous expliciterons les raisons sous-jacentes de cette subdivision du corpus dans le deuxième chapitre. C'est pourquoi il est présenté seulement en *Annexe D : Corpus développemental indirect*, rédigé à la fin de cette recherche. On peut désormais présenter le corpus développemental direct dans le tableau ci-dessous, dont on justifiera à nouveau les choix au fur et à mesure de la thèse.

Type d'outil	Éléments de corpus
Langages de programmation	*Blueprints, C*++ 14, *Unreal C++*.
Moteur de jeu	*Unreal Engine* 4.27

Tableau 8. Corpus développemental direct de la recherche

Au regard de ce corpus développemental direct, il est important de souligner que notre niveau d'analyse, sur le plan interactif du moins, se concentre essentiellement sur les moteurs de jeu en eux-mêmes, notamment *Unreal Engine*. Si nous évoquerons des API graphiques comme *OpenGL* ou *DirectX*, lors du deuxième chapitre par exemple, c'est uniquement dans l'objectif de retracer brièvement l'évolution des outils de développement. En réalité, une API comme *OpenGL* est bien trop complexe pour être traitée dans le cadre de notre

problématique, même superficiellement, et surtout pour une délimitation temporelle aussi large. Une simple analyse pourrait même s'étendre sur plusieurs dizaines de pages, puisqu'il serait nécessaire d'évoquer les méthodes de rendu 3D, la manière de convertir les données en affichage 3D ainsi qu'une infinité d'autres aspects techniques. Or, ces multiples paramètres ne sont pas pertinents dans notre analyse sur l'évolution de la création du *gameplay* sur plusieurs générations de consoles. Ils représentent *de facto* des arguments pour une problématique bien différente de la nôtre. Mais nous reviendrons sur ce point dans le prochain chapitre de notre recherche.

Par ailleurs, et afin de centrer nos multiples analyses sur des aspects pertinents liés à notre problématique, il est important de circonscrire intentionnellement notre recherche à travers plusieurs limitations. Surtout, ces limites permettent de réduire le champ de recherche de cette thèse, et d'établir une meilleure structure dans ce manuscrit pour offrir des pistes et théories plus précises. Nous pouvons ainsi énumérer trois restrictions majeures que nous imposons à notre recherche :

- ► Analyse exclusive du processus de création de jeux en 3D temps réel, excluant par conséquent la 3D hors ligne ou la 2D ;
- ► Étude de la création du *gameplay*, ou des mécanismes interactifs, uniquement dans le cadre du développement de jeux solo, ou hors ligne, et s'opposant au développement de jeux multijoueurs ou en ligne ;
- ► Analyse du développement de jeux uniquement sur les différentes consoles de salon *PlayStations*, sur ordinateur ainsi que pour les multiples moteurs de jeu *Unreal*.

Ainsi, la première restriction concerne l'ancrage dans le développement de jeux en 3D temps réel, notion que nous définirons précisément par la suite, mais qui est surtout une restriction liée à une considération pratique. En effet, nous estimons que l'analyse des outils de création jeux vidéo en 3D temps réel représente une tâche extrêmement complexe ; l'ajout d'un corpus utilisant des jeux vidéo ayant des graphismes d'un type alternatif risquerait d'augmenter substantiellement la difficulté d'une recherche déjà bien complexe. Néanmoins, la conséquence directe de cette même restriction est l'impossibilité de pouvoir appliquer nos multiples théories et *in fine* résultats de notre recherche aux outils de développement de jeux en 2D, par exemple. Nous partons ainsi du principe

que nos théories portent exclusivement sur les jeux vidéo en 3D temps réel, et celles-ci pourraient tout à fait être invalidées pour des outils de conception de jeux ayant des graphismes d'une autre nature.

En revanche, la raison de la deuxième restriction, autrement dit de l'examen du développement de jeux en solo uniquement, provient d'une mise en surbrillance très importante des jeux en ligne dans la sphère universitaire. Un phénomène qui semble éclipser les jeux en solo, renforçant l'idée que la plupart des recherches semblent converger vers de multiples questionnements sur les jeux en ligne. Une étude de 2016 par Samuel Coavoux, Manuel Boutet et Vinciane Zabban va mettre en exergue cette même propension des « articles [scientifiques] (...) [à être] focalisés sur une minorité de jeux : les jeux en ligne massivement multijoueurs » (Coavoux, Boutet et Zabban 2016 TDR)[103]. De surcroît, nous estimons que les jeux en ligne doivent faire l'objet d'une étude à part entière plutôt qu'être traités comme étant des fonctionnalités accessoires dans le développement. En clair, les éventuelles fonctionnalités en ligne, fonctionnelles ou non des outils et des jeux, seront intégralement ignorées.

Enfin, la troisième restriction, également d'ordre pratique, permet de limiter raisonnablement la quantité de nos analyses. Ainsi, nous pensons qu'il est préférable de se concentrer principalement sur une seule console de salon par génération, et notre choix se porte sur les différentes *PlayStation.* Ces consoles présentent l'avantage d'avoir un système de numérotation itératif, facilement compréhensible pour un néophyte dans les jeux vidéo. Notre restriction est également similaire pour les moteurs de jeu, où notre choix d'étude se porte surtout sur l'analyse des différentes versions du moteur *de jeu Unreal Engine*, possédant un système de numérotation itératif et simple, qui est de surcroît présent dans notre corpus développemental direct. Naturellement, cette troisième restriction signifie simplement une focalisation sur les différents éléments mentionnés, et non pas que nous allons complétement éluder les autres consoles ou moteurs. Nous évoquerons d'autres outils de conceptions ou

103. « [scientific] papers (...) [to be] focused on a minority of games: massively multiplayer online games » (Coavoux, Boutet et Zabban 2016).

consoles autant que nécessaire, si cela permet de mieux répondre à notre problématique, voire de nuancer certains arguments.

Au-delà du corpus développemental, et après avoir présenté les deux limitations de notre étude, nous pouvons désormais présenter le deuxième corpus : le corpus vidéoludique. Celui-ci est constitué de plusieurs jeux vidéo commercialisés au moins depuis les années 1980, voire 1995 pour la plupart des cas, et déterminés seulement au fil des recherches. La sélection des jeux en question est basée sur deux critères mutuellement exclusifs et pertinents au regard de notre problématique : soit une disponibilité d'informations intéressantes sur le processus de développement de ces mêmes jeux, soit des jeux dont certaines informations sur le *gameplay* pourraient contribuer de manière substantielle à la résolution de notre problématique. Ainsi, nous utilisons pour le premier cas des sources de nature différente, notamment des ouvrages, magazines ou documentaires, pour accéder aux informations sur le développement des jeux en question. Néanmoins, l'existence de ces mêmes informations étant par essence imprévisible, il nous semble donc impossible de présenter en introduction un corpus précis avant d'avoir achevé la thèse, contrairement au corpus développemental direct. La liste de l'ensemble des éléments, rédigée après avoir clôturé la recherche, figure donc en *Annexe B : Nouveau corpus vidéoludique*.

Dès lors, on pourrait également s'interroger sur la pertinence de ce choix et se demander s'il ne serait pas préférable de concevoir un corpus fermé et clairement défini, en sélectionnant une liste bien précise de jeux. D'abord, la constitution d'un tel corpus pourrait s'avérer extrêmement complexe dans un cadre diachronique, surtout au regard du nombre pléthorique de jeux publiés depuis les années 1990. Outre l'exemple que nous avons déjà fourni d'au moins 50 000 jeux disponibles sur la plate-forme *Steam*, signalons grâce à Scullion que la troisième génération de consoles possède près de « 714 jeux sous licence officielle pour la Nintendo Entertainment System » (Scullion 2019, 12 TDR) [104]. Mark Jones illustre également cette abondance avec l'existence de « plus de 15 000 titres au total

104. « 714 officially licensed games for the Nintendo Entertainment System » (Scullion 2019, 12).

pour la PS1 et la PS2 » (Jones 2006 TDR)[105]. Il est toutefois possible de rétorquer qu'un compromis pourrait être envisageable : peut-être serait-il possible d'isoler un échantillon pertinent et potentiellement représentatif de l'ensemble des jeux vidéo ?

En réalité, cette même interrogation taraudait déjà les méandres de notre réflexion scientifique, nous conduisant ainsi à tester celle-ci au début de cette recherche ; elle ne s'est pourtant pas avérée fructueuse. En effet, au départ, nous avions établi un corpus vidéoludique de vingt jeux par génération. Celui-ci, disponible en *Annexe E : Ancien corpus vidéoludique*, était exclusivement basé sur le nombre de ventes inscrit sur la plate-forme *VGChartz*. Un même site qui répertorie des données publiques, recueillies via une méthodologie d'estimation des ventes comportant, d'après les informations de Brett Walton, « dix procédés différents (...) disponibles pour collecter les données (la plupart de manière automatisée) (...) [parce que] VGChartz (...) n'a pas un accès direct aux données de vente des principaux commerçants » (Walton 2010 TDR)[106]. Nous nous éloignons cependant d'autres sources mentionnées par Simon Carless, notamment « la plupart des classements de jeux dans le monde ((...) NPD (...), *Media Create* (...) *Enterbrain* (...) et *Chart-Track* (...)) [qui] sont compilés par extrapolation à partir des chiffres de vente fournis par les commerçants » (Carless 2008 TDR)[107]. Ces dernières étaient, et sont encore, inaccessibles publiquement, sauf à travers quelques publications journalistiques indirectes.

Néanmoins, au regard de nos premières analyses, il nous semblait impossible d'utiliser l'ancien corpus en question pour répondre à la problématique de manière pertinente. Les multiples jeux de ce corpus initial étaient dénués d'informations sur les outils de développement et n'étaient pas toujours utiles pour notre problématique. Ainsi, avec cet ancien corpus, il était seulement possible d'élaborer une série d'hypothèses presque spéculatives, et non pas de s'appuyer sur des données tangibles pouvant répondre à la problématique de

105. « more than 15,000 titles for the PS1 and PS2 combined » (Jones 2006).
106. « ten different processes (...) available for data collection (most of them automated) (...) [because] VGChartz (...) doesn't have direct access to sell-through data from major retailers » (Walton 2010).
107. « most worldwide game charts ((...) NPD (...), Media Create (...) Enterbrain (...) and Chart-Track (...)) [which] are compiled by extrapolation from sales figures provided by retailers » (Carless 2008).

manière adéquate. C'est d'ailleurs cette raison qui a conduit à l'élaboration de notre nouvelle approche.

Par conséquent, nous pouvons désormais clarifier ce qui était perceptible en filigrane : notre recherche se concentre principalement sur l'exploration des possibilités offertes par les outils de conception de jeux, plutôt que sur les mécanismes interactifs déjà présents dans un jeu vidéo commercialisé. Concrètement, nos analyses portent, par exemple, sur les possibilités offertes par le moteur *Unreal Engine*, et comment celui-ci peut ou a été exploité pour créer des mécanismes interactifs. Et nous éluderons, la plupart du temps, des analyse du *gameplay* existant dans des jeux comme *Tekken 8* (Bandai Namco 2024, Multi) ou *Grand Theft Auto V* (Rockstar North 2013, Multi), qui ne répondent pas à notre problématique.

Au-delà de l'analyse de création des mécanismes interactifs dans le développement de jeux vidéo 3D, il est important de spécifier la manière dont nous allons traiter la typologie des genres vidéoludiques dans notre recherche : nous définirons ceux-ci seulement à l'aune de leur existence dans les outils de développement de jeux. En clair, nous ne présenterons pas une description exhaustive de chaque genre vidéoludique, et notre objectif est uniquement de pouvoir repérer les genres eux-mêmes, puis de les définir le cas échéant, notamment à partir des multiples listes en *Annexe C : Présence des genres vidéoludiques*. Un choix que nous pouvons justifier par de multiples arguments majeurs que nous allons exposer. D'abord, nous estimons qu'une définition des genres, même succincte, serait indéniablement hors sujet dans cette recherche qui implique d'étudier les outils techniques plutôt que des définitions. Ainsi, nous risquerions *de facto* d'égarer notre propos si celui-ci s'immisce dans des considérations lexicales qui, de surcroît, ne pourraient aucunement répondre à notre problématique. Ajoutons que notre recherche est centrée sur les éventuelles limitations technologiques des *outils* de développement, et non pas sur la définition ou la variation des genres vidéoludique en eux-mêmes.

Concrètement, nous vérifierons tout au long de notre recherche si les genres existent dans un outil comme un moteur de jeu ou des langages de programmation. En d'autres termes, si nous pouvons trouver une typologie ou des éléments qui vont permettre de différencier un genre *RPG* d'un genre *combat* à travers la programmation du *gameplay* sur *Unreal Engine*. De surcroît, nous

analyserons le *C++* et le *Blueprints*, pour voir si cette typologie des genres vidéoludiques est présente et, si c'est le cas, sous quelle forme. En revanche, la question de savoir si la série *Tekken* (*ab*. 1994, Arcade) relève du genre *combat* ou le jeu *Metal Gear Solid* (Konami 1998, PS1) pourrait être inclus dans le genre *action-infiltration* est en dehors de notre problématique.

On pourrait alors rétorquer qu'il serait difficile de vérifier si les genres existent dans les outils de développement si nous ne les définissons pas en amont ; une nomenclature avec des critères et explications précises sur certains genres pourrait donc être pertinente selon cette même hypothèse. Or, une telle argumentation nous semblerait fallacieuse à plusieurs égards. D'abord, l'objectif de notre recherche est *de facto* axé sur l'étude des outils de développement de jeux 3D pour mettre au jour d'éventuelles limitations qui peuvent structurer, et *in fine* limiter, le *gameplay*. Nous envisageons donc notre recherche en analysant les outils eux-mêmes pour identifier les éventuelles limitations qui existent, et vérifier si le genre vidéoludique correspond à l'une des limitations en question.

De surcroît, une définition en amont des types de jeux vidéo risquerait d'induire un potentiel biais de recherche, où nous analyserions les outils seulement à l'aune de genres préétablis via des descriptifs sémantiques, plutôt qu'une analyse générale des limitations pour éventuellement repérer des genres. Un prisme aussi étroit risquerait de compromettre *in fine* la scientificité des résultats de notre recherche, puisque le point de départ ne serait plus une hypothèse, mais plutôt la classification elle-même des genres. Plus encore, lors de l'élaboration de l'ancien corpus vidéoludique mentionné en amont, nous avions bel et bien procédé à la définition d'une dizaine de genres dans les premiers temps de notre recherche. Ainsi, nous avions utilisé ces définitions pour analyser les phases de *gameplay* des jeux commerciaux et extraire les caractéristiques des genres vidéoludiques. Cependant, nous avons constaté que ces analyses n'apportaient aucune pertinence pour notre problématique, et qu'elles relevaient d'une simple collection d'hypothèses presque spéculatives sur le processus de création de jeux. En d'autres termes, elles ne formulaient pas de réponses, partielles ou partiales, dans le cadre de notre recherche.

En complément, nous avions également constaté que les éventuels débats provoqués par l'exposition des multiples définitions des genres vidéoludiques orientent inexorablement la thèse vers un hors sujet au regard de l'aspect

technologique de notre problématique. Or, notre objectif est de vérifier les limitations des outils de création de jeux, et non pas d'analyser les fluctuations sémantiques ou lexicales des genres vidéoludiques. Nous considérons surtout que l'analyse des genres plutôt que de la technologie elle-même relève d'une méthodologie d'analyse distincte, propre aux *Game Studies* qui est *in fine* différente de notre approche sémio-informatique inédite. Et, justement, le contexte d'élaboration de notre approche était nécessaire pour se différencier des approches adoptées par les premières analyses scientifiques des jeux vidéo, dont les *Game Studies*. C'est pourquoi il est important d'explorer plus en détails cette scientifisation du jeu vidéo dans l'introduction de notre prochaine section.

1-2. SCIENTIFISATION DES JEUX VIDÉO

Depuis la commercialisation des premières bornes d'arcade jusqu'à notre ère technologique actuelle, examiner l'essence et la complexité des univers virtuels sous un prisme universitaire paraît, de prime abord, une tâche encore difficultueuse, voire paradoxale. De surcroît, malgré de multiples métamorphoses du paysage vidéoludique depuis sa genèse, celui-ci reste encore imprégné de clichés récalcitrants. Il semblerait même qu'il puisse, d'après les constatations de Jesper Juul, « pour une personne instruite, littéralement (...) [envoyer] des signaux d'une culture inférieure, de divertissement et d'insignifiance » (Juul 2000 TDR) [108] . Des idées préconçues qui ciblent, potentiellement, aussi bien des joueurs que des développeurs de jeux vidéo. Pourtant, la nature intrinsèque des œuvres vidéoludiques est à mille lieues de tels stéréotypes, comme souligné par Mike McShaffry et David Graham, puisque celle-ci est « extrêmement complexe, et leur création impliquent une combinaison unique d'art et de science » (McShaffry et Graham 2013, VIII TDR)[109]. Des constatations partagées par d'autres protagonistes de l'industrie, mettant en évidence le fait que, d'après Alf Inge Wang et Njål Nordmark, « le développement de jeux AAA (des titres de jeux majeurs) requiert une multitude de compétences aussi bien en informatique (...) que dans d'autres disciplines comme l'art, le *game design* et l'audio/ la musique » (Wang et Nordmark 2015, 273 TDR)[110]. Nous pourrions considérablement élargir cette liste de disciplines, car le développement de jeux implique également des connaissances historiques, culturelles, sémiotiques, et bien d'autres encore.

En revanche, une analyse scientifique et méthodique doit bien s'interroger sur les différents critères définitoires de ce même objet ludique : faut-il considérer les jeux vidéo comme étant des jeux (au sens large), des logiciels informatiques, des univers fictifs scénarisés ou encore un media de masse interactif ? Pour

108. « to an educated person, literally (...) [send] signals of low culture, fun and insignificance » (Juul 2000).

109 . « insanely complex, and their creation involves a unique combination of art and science » (McShaffry et Graham 2013, VIII).

110. « the development of AAA games (major game titles) requires a multitude of computer science skills (...) as well as other disciplines as art, game design, and audio/ music » (Wang et Nordmark 2015, 273).

mieux appréhender ces multiples propositions, il est nécessaire de recontextualiser la chronologie de l'étude scientifique du jeu vidéo dans cette deuxième section de notre introduction. Celle-ci permettra *in fine* de mettre en surbrillance les multiples raisons du choix de définir, dans notre étude, les jeux comme étant des logiciels informatiques avant tout. Une définition qui semble être, à plusieurs égards, incompatible avec certaines approches dans l'étude du jeu vidéo, notamment celle nommée *Game Studies*, comme nous le verrons dans la prochaine sous-section. C'est la raison pour laquelle nous présenterons ensuite les deux domaines qui traversent cette thèse, l'informatique et la sémiologie, dans les prochaines sous-sections. Puis nous terminerons cette introduction par le plan de la thèse.

D'abord, eu égard à la pluralité des propositions des définitions exposées *supra*, les nombreuses recherches qui s'inscrivent dans ce même champ des *Game Studies* catégorisent habituellement le jeu vidéo comme étant, d'après Gauvain Leconte, « un sous-ensemble de la classe des jeux (...) [ou] une extension de la classe des jeux à d'autres objets » (Leconte 2019, 7). Cette filiation implique inéluctablement de reprendre, d'après les propos de Maude Bonenfant, l'héritage des premières recherches sur le « jeu, [qui,] en tant que concept, n'est étudié que depuis le XX^{e} siècle » (Bonenfant 2010, 1). Et ce, principalement grâce aux théorisations historiques des « précurseurs, Johan Huizinga (1938) et Roger Caillois (1958) [qui] ont chacun donné une définition basée sur l'énumération de critères inhérents à tous les jeux » (*ibid.*). A fortiori, cet ancrage sempiternel des œuvres vidéoludique dans l'univers général du jeu semble impliquer, d'après Laurent Di Filippo, que lesdites « théories (...) font office de "passage obligé" pour les chercheurs sur le jeu » (Di Filippo 2014, 282).

Néanmoins, même si ces théories sont souvent utilisées, Di Filippo constate que « beaucoup (...) se sont attachés à les présenter afin de les critiquer » (*ibid.*), sans toutefois remettre en cause la filiation systématique des jeux vidéo au jeu. Plus encore, la plupart des auteurs ancrés dans les *Game Studies* vont mettre en exergue cette volonté d'étudier, comme indiqué par Katie Salen et Eric Zimmerman, « toutes sortes de jeux : des jeux de stratégie sur papier et des jeux de tir à la première personne, des jeux de société et des jeux de hasard

étincelants » (Salen et Zimmerman 2004, 1 TDR)[111]. Sur ce point, Aarseth indique également que le fait « d'exclure l'étude des jeux D&D (ainsi que des jeux de guerre, de rôle, de société (...) et des jeux d'argent) du (...) champ [des *Game Studies*] n'a aucun sens sur le plan académique et scientifique » (Aarseth 2017 TDR)[112]. Une même filiation au domaine général du jeu qui est également présentée dans l'ouvrage de Crawford (1984) mentionné en première partie. Celui-ci définissait, en l'occurrence, « quatre catégories majeures de jeux : les jeux de plateaux, les jeux de cartes, les jeux de sport et les jeux d'ordinateur » (Crawford 1984, 2 TDR)[113], mettant en avant l'aspect ludique comme seul critère en commun.

De plus, nous pouvons souligner que cette même stratégie trouve également ses origines au cœur même de l'industrie vidéoludique, bien au-delà des considérations théoriques de Crawford ou des *Game Studies*. En effet, dans les années 1980, différents acteurs du marché avaient pour objectif de s'introduire subrepticement dans les foyers, constate Étienne Armand Amato, « sous l'allure anodine d'un produit de simple divertissement pour enfants et adolescents, aux États-Unis, en Europe et au Japon » (Amato 2014, 163). En outre, la crise majeure de 1983 expliquée précédemment semble avoir justifié ce prisme du ludique. Une même stratégie qui s'est avérée plutôt fructueuse, d'après Roberto Dillon, puisqu'on constate que « les jeux vidéo avaient fortement regagné leur place dans le cœur des enfants (...) mais (...) avaient encore du mal à être pleinement acceptés par une grande partie de la société » (Dillon 2011, 120 TDR)[114].

Or, nous nous trouvons en face du premier point de discordance majeur entre les *Game Studies* et notre approche novatrice : un cadre théorique imposé par ce champ spécialisé sur les jeux qui est incompatible avec notre recherche. En effet, nous avons formulé une problématique interrogeant principalement la dimension *informatique* ou *technologique* du développement de jeux vidéo 3D. De surcroît,

111. « all kinds of games: paper-based strategy games and first person shooters, board games and glitzy gambling games » (Salen et Zimmerman 2004, 1).
112. « to exclude the study of D&D (and war games, roleplay, board games, (...) and gambling games) from (...) the field [of Game Studies] does not make good, academic sense » (Aarseth 2017).
113. « four major categories of games: board games, card games, athletic games, and computer games » (Crawford 1984, 2).
114. « video games had firmly regained their spot in the hearts of kids (...), but (...) still struggled to be widely accepted by a large part of society » (Dillon 2011, 120).

les multiples systèmes de signes sémiologiques que nous allons élaborer dans cette thèse doctorale existent dans l'unique objectif de mieux appréhender ces outils informatiques, *de facto* liés aux jeux vidéo. Et, dans ce cas spécifique, nous éludons intégralement l'aspect ludique des jeux vidéo dans notre étude pour nous concentrer exclusivement sur la dimension technologique. Naturellement, nous considérons que les interrogations portant sur les aspects ludiques sont parfaitement légitimes sur le plan scientifique, et *in fine* éclairantes dans certains contextes, au-delà de notre problématique. Ainsi, il pourrait être très pertinent pour certaines études de questionner le côté ludique de la conduite dans la série *Gran Turismo* (*ab*. 1997, PS1) ou du contrôle du personnage *Lara Croft* dans les épopées virtuelles de la saga *Tomb Raider* (*ab*. 1996, Multi). Plus encore, ces considérations sur le divertissement occupent une place significative, voire capitale, dans l'industrie du jeu vidéo. Cependant, et dans le cadre de notre recherche ainsi que de notre problématique, elles relèvent vraisemblablement d'un hors sujet.

C'est pourquoi nous faisons l'hypothèse que ce même cadre théorique des *Game Studies*, autrement dit cette filiation *systématique* des jeux vidéo au domaine du jeu et du ludique, occulte potentiellement toutes les spécificités du jeu vidéo en tant que logiciel informatique. De surcroît, les potentielles théories qui sont développées dans une recherche sous le prisme des *Game Studies* sont censées, hypothétiquement, être compatibles avec l'ensemble des jeux. Autrement dit, une théorie doit prendre en considération le jeu vidéo *Super Mario Odyssey* (Nintendo 2017, Switch), le jeu de cartes du *Poker* et le jeu de société *Monopoly*[115]. Or, cela nous semble *de facto* difficilement conciliable avec la programmation ou l'infographie 3D, où les autres jeux n'ont aucune pertinence. Cette difficulté presque infrangible pourrait potentiellement expliquer pourquoi le champ des *Game Studies* avait une possibilité de se lier à la conception de jeux, selon Pedro Pinto Neves et Nelson Zagalo, et « peut-être même en influençant l'industrie (...) [mais] cette promesse n'a finalement pas été tenue » (Neves et

115. Il faut bien noter que nous faisons référence au jeu de cartes matériel *Poker* ainsi qu'au jeu de société *Monopoly*, et non pas aux adaptations vidéoludiques de ces derniers.

Zagalo 2021, 7 TDR)[116]. En clair, les multiples études relevant des *Game Studies* ne semblent pas habituellement porter sur des considérations technologiques importantes dans l'industrie vidéoludique. Un constat qui semble également être souligné par les travaux de Politowski *et al.*, indiquant une « analyse de la littérature [qui] a révélé un manque d'études académiques sur les moteurs de jeu, en particulier sur leurs caractéristiques et architectures » (Politowski *et al.* 2020, 10 TDR)[117]. La nature même du champ des *Game Studies* semble donc être la cause de cette absence d'articles et d'ouvrages consacrés au processus de développement de jeux ou aux aspects techniques.

On pourrait alors objecter qu'il existe bien un sous-domaine des *Game Studies* consacré au développement de jeux*,* nommé *Game Production Research* et mis en surbrillance par les travaux d'Henrik Engström. Or, ce même sous-domaine n'étudie pas réellement un aspect technologique du développement de jeux vidéo, puisque « la perspective technique du développement de jeux est également bien représentée (...) mais seulement sous l'angle de la gestion » (Engström 2019, 9 TDR)[118]. En d'autres termes, il n'est nullement question de programmation ou de moteurs de jeu, mais uniquement de traiter de la gestion de projet. Certes, cet aspect est fondamental dans le développement d'un jeu vidéo, mais cela ne concerne aucunement les outils informatiques qui sont au cœur de notre recherche. Nous constatons donc un hypothétique éloignement significatif des questions technologiques dans le domaine des *Game Studies*, signifiant *de facto* une incompatibilité fondamentale avec notre recherche.

Au-delà du cadre théorique des *Game Studies*, lors de la première intensification des études scientifiques sur le jeu vidéo, ce ne sont pas les spécificités ludiques qui étaient mises en exergue. En effet, il s'agissait surtout de répondre à des craintes majeures suscitées par une potentielle corrélation entre l'usage des jeux vidéo et des comportements violents. En clair, et d'après les observations de Zabban, c'était dans l'objectif de mieux répondre à ces

116. « perhaps even influencing the industry (...) [but] this promise has [in fact] not been delivered » (Neves et Zagalo 2021, 7).

117. « our literature review [which] showed a lack of academic studies about engines, especially their characteristics and architectures » (Politowski et al. 2020, 10).

118. « the technical perspective on game development is also well represented (...) but mainly from a management perspective » (Engström 2019, 9).

appréhensions que les jeux vidéo « ont commencé à se développer en tant que champs d'études, (…) [surtout] autour des années 1990 » (Zabban 2012, 141). C'est pour cette raison que, selon Sébastien Genvo, « la plupart des premiers ouvrages publiés sur le sujet l'ont été en réponse aux critiques et craintes que pouvait susciter le domaine » (Genvo 2006, 13), autrement dit pour répondre aux inquiétudes portant sur les comportements potentiellement violents évoqués.

Et cela, en particulier lorsque cette éventuelle relation fut mise en perspective avec la tendance de l'industrie vidéoludique à concevoir des jeux principalement destinés à des enfants ou adolescents. Cette même inquiétude a atteint son paroxysme, d'après Craig Anderson et Brad Bushman, lorsque de nombreux « jeux vidéo violents émergèrent dans les années 1990, avec (...) Mortal Kombat, Street Fighter, et Wolfenstein 3D » (Anderson et Bushman 2001, 354 TDR)[119]. Cette augmentation substantielle de la violence dans ces produits de divertissement entraîna un événement majeur aux États-Unis, comme rappelé par Cameron Swan : une « audience du Congrès américain en 1993 (...) avec Mortal Kombat (...) [et] le jeu FMV Night Trap » (Swan 2022 TDR)[120]. Pour le grand public, constate Patrick Klepek, ces jeux paraissaient être des « représentations réalistes de violence qui exige des figures de l'industrie de répondre aux questions – au risque de voir les jeux vidéo régulés par le gouvernement » (Klepek 2016 TDR)[121]. Autrement dit, ces multiples jeux catalysaient les craintes majeures du grand public face à l'influence, potentiellement néfaste, du jeu vidéo sur la société contemporaine.

De cette audience émergea un nouveau système de classification des œuvres vidéoludiques, indique Swan, basé sur les tranches d'âge et « lancé en 1994, l'ESRB, (...) utilisé par la plupart de l'industrie vidéoludique » (Swan 2022 TDR)[122]. Son objectif principal était de rasséréner les potentiels acheteurs en fournissant des informations plus limpides sur le contenu du jeu. Quelques années plus tard, plusieurs systèmes fortement similaires à l'ESRB ont vu le jour, d'après Leyla

119. « violent video games came of age in the 1990s, with (...) Mortal Kombat, Street Fighter, and Wolfenstein 3D » (Anderson et Bushman 2001, 354).
120. « U.S. Congressional hearing in 1993 (...) with Mortal Kombat (...) [and] the FMV game Night Trap » (Swan 2022).
121. « increasingly realistic depictions of violence to demand figures in the industry answer questions–or risk video games being regulated by the government » (Klepek 2016).
122. « launched in 1994, the ESRB, (...) used by most of the gaming industry » (Swan 2022).

Dogruel et Sven Joeckel, notamment « PEGI (...), établi en 2003 » (Dogruel et Joeckel 2013, 674 TDR)[123]. Toutefois, on note que ces classifications par des tranches d'âge semblent avoir un impact relativement faible auprès des joueurs. Un constat valable même plusieurs années après la mise en place de ces catégories. Car, au début des années 2000, Anderson et Bushman constatent que « les adolescents (...) indiquent que 90% de leurs parents ne consultent jamais la classification des jeux vidéo avant d'autoriser l'achat » (Anderson et Bushman 2001, 354 TDR) [124] . Ces systèmes ne permettaient donc pas, selon les constatations exposées, de proposer des solutions adéquates sur les questions de violence, qui demeuraient importantes dans la sphère scientifique.

Néanmoins, Celia Hodent ajoute que, si cette même violence était « le principal sujet de recherche sur les jeux vidéo, (…) les conclusions tirées de ces études ont été largement débattues » (Hodent 2021, 60 TDR)[125]. En effet, les résultats des recherches scientifiques, parfois tranchants, permettent de remarquer que « la méthodologie de certaines études a été remise en question, et quelques-unes d'entre elles ont même été rétractées en raison de défauts trop importants » (*ibid*. TDR)[126]. Et, à ce jour, il existe de nombreuses méta-études qui semblent indiquer une situation indéterminée sur une éventuelle corrélation entre comportements violents et jeu vidéo. C'est justement ce qui est exposé par une méta-étude réalisée par Aaron Drummond, James Sauer et Christopher Ferguson, où ils indiquent que « aucun consensus n'a été atteint entre les chercheurs » (Drummond, Sauer et Ferguson 2020, 2 TDR) [127]. Nonobstant cette absence d'unanimité, Maya Mathur et Tyler VanderWeele mettent en exergue une certaine constance d'effets produits par les jeux vidéo, puisque « ces méta-analyses contradictoires fournissent en réalité un consensus important en faveur d'effets

123. « PEGI (...), established in 2003 » (Dogruel et Joeckel 2013, 674).
124. « teens (...) report that 90% of their parents never check the ratings of video games before allowing their purchase » (Anderson et Bushman 2001, 354).
125. « the main focus of the research on video games, (…) the conclusions drawn from these studies have been heavily debated » (Hodent 2021, 60).
126. « the methodology of some studies [that] has been put into question, and a few of them even got retracted because they turned out to be too flawed » (Hodent 2021, 60).
127. « no consensus has been reached among scholars » (Drummond, Sauer et Ferguson 2020).

néfastes, bien que petits et constants, de jeux vidéo violents sur le comportement agressif » (Mathur et VanderWeele 2019, 707 TDR)[128].

C'est pourquoi des investigations concernant ces problématiques de violence restent importantes, voire essentielles selon Ferguson, et les « débats entre les chercheurs ayant des opinions divergentes sur ce sujet sont potentiellement positives et éclairants pour tous les participants » (Ferguson 2015, 658 TDR)[129]. Et cela non seulement sur le plan sociétal, mais également pour l'industrie vidéoludique elle-même, qui pourrait potentiellement adapter ses œuvres en conséquence. De surcroît, d'autres phénomènes tels que l'addiction occupent également une place centrale, d'après Pierpaolo Limone, Benedetta Ragni et Giusi Antonia Toto, puisque « l'addiction aux jeux vidéo est un champ de recherche qui suscite beaucoup d'intérêt et, par conséquent, connaît une croissance significative » (Limone, Ragni et Toto 2023 TDR)[130].

Quoi qu'il en soit, les interrogations soulevées par ces multiples débats presque sempiternels sur la corrélation entre les jeux vidéo et le comportement des joueurs semblent inadéquats au regard de notre problématique. Il nous paraissait néanmoins important de les aborder en introduction afin de dresser un tableau de la situation des études scientifiques sur les jeux vidéo. De surcroît, plus qu'une simple évocation de l'état des recherches universitaires, nous souhaitons désormais schématiser les types d'études qui existent au sein des *Game Studies*. Cela permettra d'illustrer de manière plus précise son impertinence pour notre étude et, *in fine,* les raisons de la conception de notre approche novatrice.

128. « these conflicting meta-analyses in fact provide considerable consensus in favor of consistent, but small, detrimental effects of violent video games on aggressive behavior » (Mathur et VanderWeele 2019, 707).

129. « debates among scholars with different views on this topic are potentially healthy and elucidating for all involved » (Ferguson 2015, 658).

130. « video gaming addiction is an area of research that has been witnessing much interest and, subsequently, growth » (Limone, Ragni et Toto 2023).

1-2-A. INCOMPATIBILITÉ DES *GAME STUDIES*

Nous avons mis en évidence une scientifisation des jeux vidéo ayant, du moins au départ, surtout tendance à explorer les potentielles conséquences négatives du jeu vidéo sur le comportement des joueurs. Par ailleurs, nous avons également constaté que le champ des *Game Studies* est particulièrement lié à des considérations presque exclusivement ludiques, mais qu'il est intrinsèquement moins porté sur la création de jeux. Un même champ de recherche scientifique qui débuta précisément, d'après les constatations de Sebastian Deterding, « au début des années 2000 (...) [lorsque] les chercheurs ont déclaré la "Première Année" des *Game Studies* » (Deterding 2016, 3 TDR)[131]. L'objectif initial était de réunir une collection d'études universitaires structurée comme étant, d'après Aarseth, un véritable « champ académique émergent, viable et international » (Aarseth 2001 TDR)[132].

Cette volonté de développer un domaine académique spécifique était une conséquence, note Alexis Blanchet, de la « situation indéterminée du jeu vidéo [à l'université qui] a amené une majorité des chercheurs à le traiter à l'aune de leurs spécialités respectives » (Blanchet 2010, 10). Ce constat souligne bien que les premières recherches des années 1990 n'étaient pas rattachées à un cadre théorique particulier. Mais le fait de créer, et de surcroît scientifiser, le champ des *Game Studies* semblait permettre de regrouper ces mêmes études de manière plus cohérente. Par ailleurs, les *Game Studies* se placent inexorablement sous l'héritage d'un autre champ scientifique, d'après Sarah Meunier, puisque nous pouvons retracer son « origine dans le mouvement anglo-saxon des *Cultural Studies* » (Meunier 2017, 379). Le fait de contextualiser l'origine des *Game Studies* permet donc de mettre en exergue, d'après David Nieborg et Joke Hermes, que la « perspective des *Cultural Studies* aide à démontrer les spécificités culturelles des jeux et de la pratique du jeu » (Nieborg et Hermes 2008, 143 TDR)[133]. Une approche qui semble *de facto* exclure des interrogations

131. « in the early 2000s (...) [when] scholars declared the "Year One" of game studies » (Deterding 2016, 3).
132. « emerging, viable, international, academic field » (Aarseth 2001).
133. « cultural studies perspective helps to argue the cultural specificity of games and gaming as practice » (Nieborg et Hermes 2008, 143).

techniques ou des considérations informatiques portant sur le développement de jeux, pourtant au cœur de notre recherche. Une situation qui contraste avec la sémiologie, que nous redéfinirons plus loin dans cette introduction et qui, dans un tel cas, permet de pouvoir intégrer harmonieusement la technologie dans les sciences du langage.

Néanmoins, le nombre de publications du champ des *Game Studies* a progressivement augmenté au fil du temps, s'ancrant dans une profonde interdisciplinarité et *in fine* une grande multiplicité des points de vue ; de nombreuses revues scientifiques consacrées aux jeux vidéo virent le jour. Et Ekaterina Galanina, Evgenii Akchelov et Elena Sakharova constatent que, grâce à cette nouvelle perspective de légitimation du discours scientifique, les « jeux vidéo ne sont plus un simple divertissement contemporain » (Galanina, Akchelov et Sakharova 2020, 41 TDR) [134] . Désormais, ils deviennent éminemment importants dans notre société contemporaine. C'est dans ce contexte qu'on peut commencer à percevoir l'influence majeure des œuvres vidéoludiques, d'après Frans Mäyrä, et c'est pourquoi « l'impact des jeux sur la culture ou la société, la technologie ou l'économie, [qui] ne pouvait plus être ignorée » (Mäyrä 2009, 3 TDR) [135] . Certaines méta-études permettent également d'illustrer de manière limpide la croissance des publications vidéoludiques dans la sphère universitaire, en répertoriant une kyrielle d'articles et d'ouvrages scientifiques. Par exemple, une première méta-étude de Melcer *et al.* fait état, entre 2000 et 2014, de précisément « 8,207 articles scientifiques dans le domaine de la recherche du jeu » (Melcer *et al.* 2015, 3 TDR)[136]. Une autre méta-étude de Paul Martin permet de comptabiliser au moins « 24,128 documents (...) publiés (...) [entre] 1966 (...) [et] 2016 » (Martin 2018 TDR)[137].

Cependant, loin d'être des publications réellement unifiées, elles coexistent de manière plutôt disparate et hétéroclite dans le champ de recherche des *Game Studies*. Plus encore, l'aspect méthodologique des *Game Studies* est

134. « Video games are not just present-day entertainment. » (Galanina, Akchelov et Sakharova 2020).
135. « the impact of games in culture or society, for technology or economics, [which] could no longer be ignored » (Mäyrä 2009, 3).
136. « 8,207 papers from the games research field » (Melcer *et al.* 2015, 3).
137. « 24,128 documents (...) published (...) [between] 1966 (...) [and] 2016 » (Martin 2018).

difficilement définissable, d'après Sybille Lammes, notamment parce que les « Game Studies telles qu'elles sont pratiquées dans les sciences humaines manquent encore de méthodologies claires » (Lammes 2007, 25 TDR)[138]. De plus, Aarseth indique que ce champ semble intrinsèquement désordonné puisque, en réalité, il « n'y a jamais eu, et il n'y aura jamais de discipline ou de -ologie, autrement dit, un domaine cohérent défini par un ensemble unique de méthodes, des objets empiriques et des problématiques ou motivations de recherche » (Aarseth 2015 TDR)[139]. Cette absence d'ordre représenterait toutefois un atout significatif dans le milieu scientifique, selon Simon Egenfeldt-Nielsen, Jonas Smith et Susana Tosca puisque les « jeux peuvent être abordés via de nombreuses perspectives académiques et en utilisant différentes méthodologies » (Egenfeldt-Nielsen, Smith et Tosca 2015, 10 TDR)[140]. De surcroît, l'évocation même du concept d'école de pensée dans les *Game Studies*, par exemple, semblerait être, au mieux, « une exagération, puisque ces classifications ne s'identifient pas eux-mêmes comme étant des groupes, ni même comme des écoles » (*ibid.*, 12 TDR)[141]. En substance, on se retrouve d'emblée face à une difficulté apparente d'ordonner ou de structurer les *Game Studies*.

Même s'il n'existe pas d'école de pensée ou de mouvements, nous pouvons malgré cela nuancer cette manière de percevoir les *Game Studies*. En effet, on peut surtout mettre au jour des tendances générales et des schématisations des types d'analyses. Naturellement, il n'est nullement question de circonscrire un champ aussi disparate et immense à travers une description simpliste de quelques pages. Nous risquerions par une telle démarche d'en éluder une partie majeure. Nous souhaitons plutôt observer les tendances générales ; les manières habituelles de traiter des jeux vidéo. De surcroît, une telle schématisation des *Game Studies* permet surtout de mieux expliciter les raisons de notre

138. « Game-studies as practiced within the humanities still lacks clear methodologies » (Lammes 2007, 25).

139. « never was, and never will be, a discipline or -ology, that is, a coherent domain defined by a single set of methods, empirical objects, and research questions and motivations » (Aarseth 2015).

140. « games can be approached from a wide range of academic perspectives and by employing a number of different methodologies » (Egenfeldt-Nielsen, Smith et Tosca 2015, 10).

141. « an overstatement, as these groupings do not usually self-identify themselves as groups, nor indeed as schools » (Egenfeldt-Nielsen, Smith et Tosca 2015, 12).

éloignement de ce champ pour ancrer notre recherche en informatique et en sémiologie. Dans cette optique, nous basons notre schématisation des *Game Studies* sur celle proposée par les travaux d'Egenfeldt-Nielsen, Smith et Tosca (2015, 11). Celle-ci subdivise les types et méthodologies d'analyses utilisées dans les *Game Studies* en cinq parties distinctes que nous présentons dans le tableau ci-dessous.

Type d'analyse	Méthodologie
Jeu	Analyse textuelle
Joueur	Observations, entretiens, enquêtes
Culture	Entretiens, analyses textuelles
Ontologie	Enquête philosophique
Métriques	Analyse statistiques de données

Tableau 9. Méthodologies des *Game Studies* (Egenfeldt-Nielsen *et al.* 2015, 11)

De cette schématisation générale, nous pouvons d'emblée réaffirmer que le champ des *Game Studies* semble particulièrement éloigné des préoccupations d'une étude portant sur le développement de jeux vidéo, annihilant tout intérêt pour notre recherche. Néanmoins, nous allons procéder à une observation plus approfondie de chaque type d'analyses schématiques exposés ci-dessus.

Ainsi, le premier est celui du **jeu**, comme défini par les travaux d'Egenfeldt-Nielsen, Smith et Tosca, où « un ou plusieurs jeux particuliers sont soumis à une analyse, l'objectif étant d'examiner les jeux en eux-mêmes et de dire quelque chose à propos de leur structure » (*ibid.*, 11 TDR)[142]. Dans ce cas particulier, Liège Game Lab ajoute que « le jeu (vidéo) est considéré comme un objet autonome, de la même façon que le structuralisme a pu définir le texte (...) et comment il se rend compréhensible et jouable » (Liège Game Lab 2019, 15). Néanmoins, ce premier type d'analyses examine les jeux exclusivement sous le prisme du joueur, à travers des éléments tels que « la récompense des joueurs, (...) la représentation du joueur dans le monde du jeu, et ainsi de suite » (Egenfeldt-

142. « one or more particular games are subjected to analysis, [and] the point is to look at games in themselves and say something about their structure » (Egenfeldt-Nielsen, Smith et Tosca 2015, 11).

Nielsen, Smith et Tosca 2015, 11 TDR)[143]. Or, notre recherche ne pourrait aucunement s'arc-bouter sur celui-ci, puisque nous étudions le processus de développement de jeux vidéo, et plus encore celui de la création des mécanismes interactifs. Autrement dit, notre questionnement sur l'interactivité comprend certes les *interactions* elles-mêmes, mais surtout le contexte de création du jeu en question, principalement lié à la programmation et à l'infographie. De surcroît, nous ne nous intéressons jamais à la perception du joueur ou à son expérience, car cela relève d'une problématique substantiellement différente de la nôtre. Ainsi, le point principal abordé par notre problématique est donc hors du cadre délimité par ce type d'analyse, centré, d'après Egenfeldt-Nielsen, Smith et Tosca, sur « l'expérience du joueur telle qu'élaborée par le *game designer* » (*ibid*. TDR)[144]. Il s'agit surtout d'un modèle d'analyse souvent « choisi par ceux ayant une formation en littérature comparée ou dans d'autres discipline esthétiques » (*ibid*. TDR)[145] plutôt que par des recherches techniques. En substance, notre recherche s'appuie essentiellement sur l'analyse des outils de création de jeux plutôt que sur l'interactivité des jeux commerciaux. Lorsque nous mentionnerons, à plusieurs reprises, certaines caractéristiques des jeux commerciaux, c'est seulement dans l'objectif de les comparer à des outils de développement, sans un quelconque rapport avec la perception des joueurs.

Au regard du deuxième type d'analyse exposé par Egenfeldt-Nielsen, Smith et Tosca, celui des **joueurs**, nous constatons qu'il est particulièrement utilisé par « les sociologues et les ethnographes » (*ibid*. TDR)[146], et semble relever de la dimension sociale liée aux joueurs de jeux vidéo. Celui-ci se concentre principalement, d'après Liège Game Lab, sur des « approches sociales, [où] il faut citer Jacques Henriot, (...) cherchant à définir le jeu par les joueurs plutôt que par le dispositif » (Liège Game Lab 2019, 13). Une analyse qui permet *in fine* d'élaborer une description plus approfondie du contexte social, en mettant particulièrement l'accent sur la manière dont « les joueurs utilisent les jeux

143. « player reward, (...) player representation in the gameworld and so on » (Egenfeldt-Nielsen, Smith et Tosca 2015, 11).
144. « the player experience which the game designer aims for » (Egenfeldt-Nielsen, Smith et Tosca 2015, 11).
145. « often the type of analysis chosen by those with a background in comparative literature or other aesthetic disciplines » (Egenfeldt-Nielsen, Smith et Tosca 2015, 11).
146. « sociologists and ethnographers » (Egenfeldt-Nielsen, Smith et Tosca 2015, 11).

comme étant un type de media ou un espace social » (Egenfeldt-Nielsen, Smith et Tosca 2015, 11 TDR)[147]. Nous pourrions transposer cette méthode à l'étude du contexte social de développement de jeux, mais cela ne répond aucunement à notre problématique. Car nous étudions les outils de création en eux-mêmes, et non pas le contexte social, voire sociétal de création des moteurs, langages ou logiciels de conception. Ainsi, ce deuxième type d'analyses est totalement inadéquat pour notre recherche.

En outre, le troisième modèle d'analyse des travaux d'Egenfeldt-Nielsen, Smith et Tosca porte sur la **culture** vidéoludique, autrement dit le fait de pouvoir, d'après Liège Game Lab, « étudier ce qui se déroule avant, après ou à côté (...) comment le médium est raconté, critiqué, détourné ou célébré que comment il est joué » (Liège Game Lab 2019, 15). Pour autant, si nous pourrions trouver certaines similitudes avec notre recherche, les objectifs sont substantiellement différents. En effet, nous nous appuyons parfois sur des éléments discursifs provenant de plusieurs développeurs, mais seulement dans l'objectif d'avoir accès à des aspects historiques du développement de jeux et des informations sur d'anciens outils. En revanche, notre propos ne porte aucunement sur la culture du développement de jeux vidéo, ni sur la manière dont le développement de jeux est perçu par des journalistes ou des joueurs. Cette étude se focalise plutôt exclusivement sur la nature intrinsèque des outils de la création des jeux vidéo.

Par ailleurs, le quatrième type d'analyses indiqué par les travaux de des travaux d'Egenfeldt-Nielsen, Smith et Tosca est celui de **l'ontologie**, dédié à examiner « les fondements philosophiques des jeux (...) pour présenter des énoncés généraux qui s'appliquent à la totalité des jeux » (Egenfeldt-Nielsen, Smith et Tosca 2015, 11 TDR) [148] . Mais la possibilité de construire un discours philosophique général sur le jeu vidéo semble, dans ce cas, être applicable uniquement aux jeux vidéo commercialisés, éludant les éventuels aspects techniques. Or, par essence, cette conception philosophique sur les jeux vidéo

147. « players use games as a type of media or as a social space » (Egenfeldt-Nielsen, Smith et Tosca 2015, 11).

148. « the philosophical foundations of games (...) to present general statements that apply to all games » (Egenfeldt-Nielsen, Smith et Tosca 2015, 11).

n'a aucun aspect lié à l'informatique et relève *de facto* d'un discours inadaptable à notre problématique.

Enfin, le cinquième et dernier type d'analyse des travaux d'Egenfeldt-Nielsen, Smith et Tosca, celui des **métriques**, se « focalise sur (...) des mesures quantitatives du comportement des joueurs » (Egenfeldt-Nielsen, Smith et Tosca 2015, 12 TDR)[149]. Si nous pourrions considérer qu'il s'agit, de prime abord, d'une analyse liée au développement de jeux, celle-ci est cependant très restrictive dans sa définition. En effet, elle concerne explicitement « la relation entre le *game design* et le comportement (ou le sentiment) du joueur, par exemple pour aider les développeurs à améliorer l'expérience des joueurs » (*ibid*. TDR)[150]. En clair, il s'agit d'une approche statistique qui va analyser de manière potentiellement informatisée le comportement des joueurs, pour établir des conclusions ou élaborer des théories par la suite. En revanche, il n'est nullement question de s'intéresser aux outils de développement eux-mêmes ni à la manière de concevoir des mécanismes interactifs.

Ainsi, à la lumière des multiples types d'analyses schématiques des *Game Studies* énoncés ci-dessus, et en plus des autres points évoqués précédemment, nous considérons que ce champ n'est pas pertinent pour notre recherche. De surcroît, son éloignement intrinsèque des considérations technologiques entrave la possibilité de traiter de manière exhaustive des multiples aspects de notre problématique. Celui-ci semble surtout pertinent pour étudier les aspects ludiques des jeux vidéo, éléments que nous éludons explicitement. C'est donc ce qui explique l'ancrage de notre recherche dans le domaine de l'informatique et de la sémiologie plutôt que dans le champ des *Game Studies*.

On pourrait alors objecter que cette nature intrinsèque des *Game Studies* n'empêche aucunement d'intégrer des données technologiques ou informatiques, et que ce champ n'est en rien incompatible avec notre recherche. De surcroît, on pourrait invoquer son aspect disparate, comme souligné par Aarseth (2015) précédemment. Or, une telle démarche nous paraît extrêmement

149. « focusing on (...) quantitative measures of player behavior » (Egenfeldt-Nielsen, Smith et Tosca 2015, 12).

150. « the relationship between game design and player behavior (or sentiment), for instance, to help developers improve the player experience » (Egenfeldt-Nielsen, Smith et Tosca 2015, 12).

peu rigoureuse, car les *Game Studies* n'ont aucunement pour objectif de faire une étude technologique des jeux vidéo. Un champ qui ne pourrait jamais être confondu ni assimilé à l'informatique, et qui ne peut être utilisé pour des analyses techniques. Surtout, notre objectif est justement d'intégrer une rigueur importante dans l'analyse technologique portant sur le développement de logiciels ; une *obligation* d'étudier les multiples aspects techniques, et cela tout au long de notre recherche. Nous souhaitons *in fine* replacer le cadre théorique de l'informatique au cœur du questionnement scientifique sur les jeux vidéo, plutôt que proposer des analyses technologiques comme étant anecdotiques.

En résumé, cet intérêt pour une recherche axée sur le développement de jeux vidéo nous semble donc difficilement conciliable avec les caractéristiques schématiques des *Game Studies*. De surcroît, le cadre théorique de ce champ ne semble aucunement être pertinent pour répondre à notre problématique. *A contrario*, ceux de l'informatique et de la sémiotique, particulièrement vastes et abondants, permettent de formuler des réponses et pistes éclairantes pour notre recherche. C'est la raison pour laquelle il est désormais important de mieux définir et circonscrire l'informatique dans notre recherche, au cœur de notre prochaine sous-section.

1-2-B. INFORMATIQUE ET DÉVELOPPEMENT DE JEUX

Au regard des incompatibilités avec les *Game Studies* évoquées en amont, il est fondamental de définir le rôle majeur de l'informatique dans notre recherche. Néanmoins, mener une étude dans ce domaine aussi gargantuesque et diversifié nécessite de définir précisément les multiples spécialités que nous allons évoquer à l'aune de notre problématique. Dans cette optique, notre thèse traite de différentes facettes du développement de jeux vidéo en 3D temps réel ; nous pouvons principalement distinguer un sous-domaine pertinent de l'informatique pour notre recherche : le développement de logiciels, et particulièrement la programmation informatique. En substance, cela implique plusieurs composantes à analyser, que nous détaillerons tout au long de la thèse : de multiples langages de programmation, des moteurs de jeu ainsi que, en filigrane, des logiciels d'infographie. Nonobstant cette haute technicité présente tout au long de la recherche, nous expliquerons et vulgariserons la plupart des notions techniques, dans l'objectif de rendre cette thèse accessible. Car il convient de rappeler que cette étude résulte d'un alliage de deux disciplines naturellement éloignées, en l'occurrence, la sémiologie et l'informatique ; un manque de connaissance du lecteur dans l'une des deux disciplines pourrait entraver la compréhension globale de la thèse, d'où la nécessité de redéfinir la plupart des termes techniques.

Désormais, pour mieux appréhender l'informatique, on peut d'abord s'appuyer sur Abraham Silberschatz, Peter Galvin et Greg Gagne, qui vont diviser un « ordinateur (...) en quatre composantes : le *matériel*, le *système d'exploitation*, les *programmes informatiques*, et les *utilisateurs* » (Silberschatz, Galvin et Gagne 2008, 4 TDR)[151]. Cependant, nous éludons à la fois le système d'exploitation et l'utilisateur, deux aspects en dehors de notre problématique ; nous focalisons notre étude principalement sur les outils en eux-mêmes, pour vérifier s'ils sont limités par les genres vidéoludiques, et non pas sur la réception de l'interactivité par les utilisateurs, ici les joueurs. Certes, nous pourrions considérer l'utilisateur comme étant plutôt un développeur de jeu ; mais cela reste néanmoins éloigné

151. « computer system (...) into four components: the *hardware*, the *operating system*, the *application programs*, and the *users* » (Silberschatz, Galvin et Gagne 2008, 4).

de notre étude, puisque nous analysons les outils en eux-mêmes et non l'utilisation desdits outils par des développeurs. C'est pourquoi nous pouvons restructurer un ordinateur en deux composantes, grâce à la définition certes plus restrictive de Liang *et al.*, mais plus pertinente pour notre recherche. Cette autre vision de l'ordinateur « inclut le *matériel* et les *logiciels* (...) le *matériel* comprend la partie visible, les éléments physiques d'un ordinateur, et les *logiciels* fournissent des instructions invisibles qui contrôlent le matériel » (Liang *et al.* 2014, 22 TDR)[152].

En revanche, nous souhaitons mettre en surbrillance un aspect fondamental d'un ordinateur, permettant justement de créer cette même distinction entre le matériel, traduit par *Hardware* en anglais, et le logiciel, ou *Software*. Il s'agit, comme le précise Matthew Justice, de la « programmabilité, (...) un élément clé différenciant un appareil qui est un ordinateur de celui qui n'en est pas un » (Justice 2021, 118 TDR)[153]. Une capacité qui permet également de considérer une console de jeux vidéo comme étant un ordinateur spécialisé. En d'autres termes, Wardyga va mettre en exergue le fait que les « ordinateurs et consoles de jeux vidéo partagent de nombreux composants en commun qui permettent à ces appareils de jouer à des jeux vidéo » (Wardyga 2019, 29 TDR)[154]. Nous commencerons à évoquer des détails techniques dès le deuxième chapitre, mais nous rappelons que la clef de voûte de notre étude est l'interactivité des jeux vidéo.

Et cela signifie, pour reprendre cette double distinction, que notre recherche s'oriente presque exclusivement vers une analyse logicielle pour vérifier s'il existe des limites liées à la typologie des genres dans la création de l'interactivité. Ainsi, l'étude des caractéristiques matérielles est relativement éloignée de nos considérations et de notre problématique, en particulier pour un spectre temporel aussi large. Certes, nous évoquerons inexorablement certains aspects matériels, surtout pour les premières générations. Mais ces points seront mis en

152. « includes *hardware* and *software* (...) *hardware* comprises the visible, physical elements of a computer, and *software* provides the invisible instructions that control the hardware » (Liang *et al.* 2014, 22).

153. « programmability, (...) a key differentiator between a device that is a computer and one that is not » (Justice 2021, 118).

154. « computers and video game consoles share many common components that allow these devices to play video games » (Wardyga 2019, 29).

surbrillance de manière superficielle, ou haut niveau pour reprendre le jargon informatique.

Nous pouvons également évoquer un autre point fondamental permettant l'interactivité dans les jeux vidéo puisque, « en pratique, la plupart des appareils informatiques ont besoin d'interagir avec le monde extérieur, et ils réalisent cela via les périphériques d'entrées/ de sortie (I/O) » (*ibid*. TDR)[155] ; dans les jeux vidéo, le périphérique majeur est la manette de jeu, en particulier pour les consoles. Néanmoins, malgré l'importance de ces outils pour rendre l'interactivité possible, nous estimons qu'ils ne sont pas directement liés à notre problématique. En effet, celle-ci questionne la nature des mécanismes interactifs plutôt que les périphériques liés à celle-ci. C'est pourquoi nous allons ignorer les autres périphériques, par exemple comme la caméra ou le casque de réalité virtuelle.

Au-delà des manettes de jeu, le concept de programmation informatique est indubitablement vaste et protéiforme. Pour esquisser une première définition, nous pouvons affirmer que la programmation est un moyen, d'après les explications de Liang *et al.*, de pouvoir « créer (...) un logiciel, qui est appelé un programme (...) [et qui] contient des instructions qui communiquent à un ordinateur (...) quoi faire » (Liang *et al*. 2014, 22 TDR)[156]. On réalise cette opération via au moins un langage de programmation, et Michael Dawson constate qu'il existe aujourd'hui « littéralement des milliers de (...) langages de programmation » (Dawson 2014, 21 TDR)[157]. Néanmoins, et tout comme les langues naturelles humaines, Michael Scott souligne *in fine* que, « parmi les nombreux [langages] qui ont été conçus, seuls quelques dizaines sont largement utilisés » (Scott 2016, 7 TDR)[158]. Afin de classer tous ces langages de programmation, la littérature informatique se base essentiellement sur leur proximité les uns par rapport aux autres. Ainsi, on retrouve au moins deux formulations de cette typologie : une première par plusieurs niveaux, directement

155. « in practice most computing devices need to interact with the outside world, and they do so through input/ output (I/O) devices » (Justice 2021, 118).
156. « to create (...) software, which is called a program (...) [and which] contains the instructions that tell a computer (...) what to do » (Liang *et al*. 2014, 22).
157. « literally thousands of (...) programming languages » (Dawson 2014, 21).
158. « of the many [languages] that have been designed, only a few dozen are widely used » (Scott 2016, 7).

liée à la proximité avec le langage machine ; une seconde qui va segmenter les langages informatiques en différentes générations.

Grâce aux travaux de J. B. Dixit, nous pouvons définir la première conception des niveaux d'un langage informatique à travers deux extrémités : « bas niveau lorsqu'ils sont proches du langage que l'ordinateur lui-même utilise—les 1 et les 0 (...) [et] haut niveau lorsqu'ils sont proches des langues que les gens utilisent, comme l'anglais » (Dixit 2011, 57 TDR)[159]. Nonobstant une apparente simplicité de cette graduation, l'échelle entre des langages informatiques bas niveau et haut niveau semble plutôt relative, car elle dépend des autres langages utilisés à titre comparatif. D'autres variantes existent également, comme le rappelle Steven Hugg, notamment une typologie qui va concevoir les langages en trois « niveaux d'abstraction – des langages de haut niveau, des langages de bas niveau et du code assembleur » (Hugg 2018, 15 TDR)[160]. Mais nous éludons les autres variantes de cette même classification, car celles-ci ne présentent aucun intérêt pour notre recherche.

Quant à la seconde typologie, elle regroupe les différents langages en cinq générations, modèle schématique que nous pouvons illustrer à travers l'ouvrage de Dixit :

- « Première génération, 1945 — *Langage machine ;*
- Deuxième génération, milieu des années 1950 — *Langage assembleur ;*
- Troisième génération, entre le milieu des années 1950 et le début des années 1960 — *Langage haut niveau (langages procéduraux et orienté objet)* : par exemple, COBOL, BASIC, C et C++ ;
- Quatrième génération, début des années 1970 — *Langage très haut niveau (langages orientés vers les problèmes)* : par exemple, SQL, Intellect, NOMAD, FOCUS ;
- Cinquième génération, début des années 1980 — Langages naturels » (Dixit 2011, 58 TDR)[161].

159. « *lower level* when they are closer to the language that the computer itself uses—the 1s and 0s (...) [and] *higher level* when they are closer to the language people use more like English » (Dixit 2011, 57).

160. « levels of abstraction – high-level languages, low-level languages, and assembly code. » (Hugg 2018, 15).

161. « • First generation, 1945—*Machine language*. • Second generation, mid-1950s—*Assembly language*. • Third generation, mid-1950s to early 1960s—*High-level languages* (procedural

On peut également souligner que Dixit mentionne les années d'apparition des langages, qui sont encore majoritairement utilisés aujourd'hui, comme on l'observera dans notre recherche. Par ailleurs, nous focaliserons notre discours tout au long de la recherche exclusivement sur les notions de haut niveau et bas niveau, plutôt que sur ces générations des langages de programmation. La classification en différents niveaux semble plus appropriée et relativement simple pour notre problématique. De surcroît, la classification des *générations* de langages est susceptible de générer de la confusion avec les *générations* de consoles.

En outre, il est important de revenir sur les multiples implications de l'emploi de l'anglais dans le domaine de l'informatique, une pratique souvent considérée comme obligatoire. En effet, Philip Guo indique bien que « la majorité des langages de programmation, des bibliothèques, de la documentation et des manuels d'utilisation sont en anglais » (Guo 2018 TDR)[162]. C'est pourquoi l'ensemble des termes techniques et du jargon en informatique est presque exclusivement en anglais. Un phénomène mis en surbrillance par Elizabeth Saint puisque, « qu'il soit question du matériel, des composants internes, des accessoires, des logiciels ou des applications, les termes de l'informatique sont tout d'abord pensés, créés et employés en anglais » (Saint 2013, 91). Ce constat justifie *de facto* notre volonté d'utiliser principalement l'anglais pour la plupart des termes techniques de notre thèse. Néanmoins, il existe certains contextes où des traductions françaises sont attestées et largement adoptées dans le milieu professionnel. Une même traduction présente aussi bien dans des logiciels de conception de jeux que des documentations rédigées par des êtres humains. Dans un tel cas, nous utiliserons naturellement ces mêmes traductions, à condition qu'elles soient compréhensibles dans le milieu professionnel et qu'elles n'engendrent pas d'éventuelles confusions. Nous serons particulièrement vigilant sur ces deux points. Car notre objectif est *in fine* de

languages and object-oriented): For example, COBOL, BASIC, C and C++. • Fourth generation, early 1970s—*Very high-level languages* (problem oriented languages): For example, SQL, Intellect, NOMAD, FOCUS. • Fifth generation, early 1980s—*Natural languages*. » (Dixit 2011, 58).

162. « the majority of programming languages, libraries, documentation, and instructional materials are in English » (Guo 2018).

privilégier à la fois la compréhension technique et la clarté de notre discours, plutôt que de traiter d'une quelconque idéologie langagière ou politique linguistique relative à un pays en particulier.

Illustrons désormais notre propos par deux exemples que nous ne définirons que dans le cœur de la thèse : *Game Engine* et *Skeletal Mesh*. Le terme anglais *Game Engine* se traduit par *moteur de jeu*, traduction possédant qu'une seule acception en français, attestée dans un grand nombre de documents techniques, logiciels et jeux vidéo. Il ne peut en aucun cas engendrer une confusion de quelque nature que ce soit, et sera donc utilisé, en français, dans cette recherche. En revanche, le concept de *Skeletal Mesh* ne possède aucune traduction courante, d'après nos recherches. Il existe seulement une traduction littérale, inexistante dans les outils de développement et, à notre connaissance, extrêmement rare : *maillage squelettique*. Cette dernière traduction peut générer de nombreuses confusions, et dans ce cas, seul son équivalent anglais sera utilisé. On pourrait alors objecter qu'il suffit d'indiquer une traduction pour en faire une convention dans notre document. Certes, mais rappelons une nouvelle fois que notre recherche n'est pas portée sur la sémantique du jargon informatique, ni sur sa traduction potentielle, mais exclusivement sur l'interactivité. On laissera donc le soin aux recherches axées sur des analyses lexicales ou sémantiques de s'interroger de manière approfondie sur les questions qui relèvent des traductions en informatique, qui sont pour notre étude un hors-sujet. Néanmoins, ce point de vue concerne exclusivement l'informatique, et nous utiliserons plutôt des termes français pour la sémiologie. Ainsi, nous traduirons la totalité des termes sémiologiques, sauf dans les cas où ceux-ci seront mélangés à des notions en informatique.

Au-delà des considérations linguistiques liées au lexique informatique, la 3D temps réel, représentant l'aspect graphique dans la création d'un jeu, constitue un élément fondamental de notre recherche. Il convient de clairement définir celle-ci puisque ses acceptions sont parfois galvaudées. En effet, il existe des tendances hypothétiquement *marketing* ou commerciales pouvant altérer la véritable signification des termes techniques, provoquant certaines confusions chez les néophytes. En l'occurrence, et on peut le constater plus encore dans le domaine des jeux vidéo et grâce à Carl Therrien, « les supports promotionnels (…) ont souvent tendance à exagérer ou ouvertement dénaturer les caractéristiques

techniques » (Therrien 2012, 13 TDR)[163]. Il existe de nombreux exemples de jeux qui évoquent certaines technologies supposément en 3D, phénomène également illustré par Therrien, notamment pour « *Cruise for a Corpse* (...), *Axelay* (...), et *Doom* (...) [qui] ont tous été promus comme étant des exploits dans les graphismes en 3D » (*ibid*. TDR)[164]. Or, la réalité sur ce même plan technique est bien différente, constate toujours Therrien, puisque « ni la manipulation des formes vectorielles, ni les redimensionnements *bitmap* du ‘‘mode 7’’, ni l’interpolation de textures "2.5-D’’ (...) ne correspondent véritablement à (...) un moteur de jeu 3D » (*ibid*. TDR)[165].

Ainsi, il est désormais possible de définir la 3D temps réel comme étant un processus qui va, d’après la société Autodesk, « convertir des modèles 3D en images en 2D (...) calculées en temps réel, de manière à donner l’impression que de nombreuses images se produisent en temps réel » (Autodesk s. d. TDR)[166]. Cela permettra *in fine*, selon l’entreprise Unity Technologies, aux « joueurs [d’] interagir avec (…) [un] jeu » (Unity Technologies s. d.-a TDR)[167]. Cela correspond bien, pour reprendre la formulation de Jason Gregory, à notre définition des « jeux [qui] sont des simulations informatiques en temps réel, dynamiques et interactives » (Gregory 2019, 525 TDR)[168].

Il s’agit donc d’un type de rendu particulier, puisque *Unity Technologies* va définir « deux principaux types de rendu 3D (...) le rendu en temps réel et le rendu *offline* ou pré-calculé » (Unity Technologies s. d.-a TDR)[169]. Ce même rendu en 3D temps réel se différencie donc d’un rendu en 3D pré-calculée ou 3D *offline*, ce dernier étant, pour reprendre les propos de William Vaughan, « utilisé pour le cinéma, des émissions, des impressions, et d’autres marchés (...) [et] offre aux

163. « promotional material (…) often overemphasizes or blatantly distorts technical features » (Therrien 2012, 13).
164. « *Cruise for a Corpse* (...), *Axelay* (...), and Doom (...) [which] have all been promoted as achievements in 3-D graphics » (Therrien 2012, 13).
165. « neither the manipulation of vector shapes, the ‘‘mode 7’’ bitmap scaling, nor the "2.5-D’’ texture interpolation (...) truly correspond to (...) a 3-D game engine » (Therrien 2012, 13).
166. « convert 3D models into 2D images (...) calculated in real time, so that it appears as though many images are occurring in real time » (Autodesk s. d.).
167. « players [to] interact with (…) [a] game » (Unity Technologies s. d.-a).
168. « games [which] are real-time, dynamic, interactive computer simulations » (Gregory 2019, 525).
169. « two major types of rendering in 3D (...) real-time and offline or pre-rendering » (Unity Technologies s. d.-a).

artistes des options de rendu d'une qualité supérieure (...), d'une complexité illimitée » (Vaughan 2012, 148 TDR)[170].

On note également que notre recherche concerne *exclusivement* la 3D temps réel, occultant également la 3D stéréoscopique. Grâce à Wardyga, nous pouvons définir que, si « le terme '3D' (…) fait référence aux graphismes ayant plusieurs côtés et une profondeur *à l'intérieur* de l'écran (...) la **3D stéréoscopique** (…) crée l'illusion d'objets qui sortent *en dehors* de l'écran » (Wardyga 2019, 39 TDR)[171]. Nous éludons également les technologies des casques de réalité augmentée, ou encore de la réalité virtuelle, pour nous concentrer exclusivement sur la 3D temps réel avec un affichage traditionnel. Enfin, Jon Peddie permet de mettre en exergue le fait que la « 3D en infographie comporte deux éléments primaires : la géométrie et l'art » (Peddie 2013, 4 TDR)[172]. Sur ce point, notre recherche se concentre exclusivement sur l'aspect géométrique, puisque les spécificités artistiques des jeux vidéo 3D ne sont pas liées à notre problématique. Concrètement, nous formulons l'hypothèse que les multiples styles artistiques utilisés dans les jeux vidéo ne pourraient aucunement fournir des pistes au regard des éventuelles limitations des mécanismes interactifs. De surcroît, l'analyse de la véracité de cette même hypothèse n'est pas directement liée à notre problématique, et pourrait constituer une problématique de recherche à elle seule. En revanche, le domaine général de la sémiologie, profondément ancré dans tous les chapitres de notre étude, est parfaitement adapté pour répondre à notre problématique. C'est pourquoi nous allons longuement présenter celui-ci dans la prochaine sous-section.

170. « used for film, broadcast, print, and other markets (...) [and] affords artists higher-quality rendering options (...) of unlimited complexity » (Vaughan 2012, 148).
171. « the term '3D' (…) refers to graphics having multiple sides and depth *within* the screen (...) **stereoscopic 3D** (…) creates the illusion of objects protruding *beyond* the screen » (Wardyga 2019, 39).
172. « 3D in computer graphics has two primary elements: geometry and art » (Peddie 2013, 4).

1-2-C. SÉMIOLOGIE ET SÉMIOTIQUES

Les multiples facettes de l'informatique que nous venons d'explorer mettent bien en évidence qu'il s'agit d'un domaine éminemment technique. Celui-ci peut donc seulement mettre en surbrillance les limitations technologiques de la conception des mécanismes interactifs. Néanmoins, et au regard de notre problématique, nous estimons que l'informatique pourrait être *de facto* limitée dans sa capacité à aller au-delà de la technicité pour formuler des *théories* plus générales. De surcroît, notre recherche nécessite une analyse profonde sur la *signification* de ces mêmes limitations, avec une mise en corrélation pertinente de la typologie des genres vidéoludiques. Pour cela, il est nécessaire d'isoler les différents *signes* de création du *gameplay*, mais également d'élaborer une éventuelle *structure* commune, voire universelle, décrivant les outils de développement de jeux vidéo. La sémiologie se révèle donc être, au regard des éléments cités, le seul domaine susceptible de pouvoir répondre à toutes ces exigences. Pour autant, il est important d'en définir dans cette sous-section son cadre théorique, sa portée générale et, plus particulièrement, son usage précis dans notre étude.

Tout d'abord, et comme le rappelle Roland Barthes, l'objectif principal de la sémiologie et, a fortiori, de notre recherche, « est de reconstituer le fonctionnement des systèmes de signification » (Barthes 1964a, 132). Mais, de manière plus générale, on peut surtout indiquer, grâce à Daniel Chandler, que cette science « s'intéresse à la manière dont les significations sont créées et comment la réalité est représentée (et effectivement construite) à travers des signes, des systèmes de signes et des processus de signification » (Chandler 2022, 2 TDR)[173]. De surcroît, ce domaine possède l'avantage d'être applicable pour l'analyse de n'importe quelle autre discipline ou objet ; Jean-Marie Klinkenberg souligne cet aspect fondamental, en indiquant bien que la « sémiologie ou sémiotique n'a pas d'objet propre (...) elle constitue une grille d'analyse particulière (...) en posant une question (…) : quel est le sens ? »

173. « semiotics is concerned with how meanings are made and how reality is represented (and indeed constructed) through signs, sign systems, and processes of signification » (Chandler 2022, 2).

(Klinkenberg 2000, ch. 1). Plus intéressant encore, cette nature intrinsèque de la sémiologie partage des similitudes avec les *Game Studies* ; plus précisément sur le fait qu'il n'existe « pas actuellement de consensus sur l'objet même de la discipline, et encore moins sur ses méthodes » (*ibid.*).

Certes, d'après Bengu Batu, on retrouve bien la ligne directrice d'une méthodologie, bien que relativement générique, qui « implique la reconstitution et reconstruction de la structure du sens à travers l'extraction, la subversion et la déconstruction » (Batu 2012, 465 TDR)[174]. Cependant, celle-ci est difficilement définissable dans une analyse concrète, comme celle du développement de jeux vidéo. C'est pourquoi notre approche sémio-informatique novatrice du Développement Logiciel Tridimensionnel est d'autant plus pertinente : elle permet d'imposer une rigueur méthodologique éminemment précise pour notre problématique, contrairement aux *Game Studies* ou aux théories sémiologiques générales.

Ensuite, sur le plan historique, et grâce à François Rastier, il est possible de distinguer deux origines différentes de la sémiologie : « la philosophie (...) depuis Locke, (...) et la linguistique depuis Saussure, Hjelmslev, Greimas » (Rastier 2001, 2). Cela a donné naissance à deux courants schématiquement distincts : une sémiotique d'origine américaine et une sémiologie européenne, qu'Anne Hénault considère comme « deux disciplines siamoises » (Hénault 2008, 11-12). Les origines distinctes de la philosophie et également de la linguistique permettent d'associer, selon Sémir Badir, à la sémiologie le domaine connexe de « la logique dans le cas de Peirce, [et de] la linguistique dans celui de Saussure » (Badir 2010, 243). Pour autant, les travaux de Badir indiquent non pas deux mais plutôt trois théoriciens « que les sémioticiens estiment être les fondateurs de leur discipline : Charles Sanders Peirce (1839–1914), Ferdinand de Saussure (1857–1913) et Louis Hjelmslev (1899–1965) » (*ibid.*). Cela étant, on considère souvent le troisième comme étant dans la lignée de Saussure, aspect souligné par Kenji Tatsukawa, puisque « Hjelmslev lui-même se considère comme le seul continuateur du maître genevois » (Tatsukawa 1995, 479).

174. « involve the reformation and reconstruction of the structure of meaning through its extraction, subversion and deconstruction » (Batu 2012, 465).

Définissons désormais plus précisément cette sémiologie européenne, fondée par Ferdinand de Saussure dans les années 1910, comme souligné par Zufferey et Moeschler, à travers le « *Cours de linguistique générale*, donné à l'université de Genève entre 1906 et 1911 » (Zufferey et Moeschler 2021, ch. 5). Pour autant, il est impératif de contextualiser le *Cours de Linguistique Générale* (désormais CLG), une œuvre qui n'est pas directement rédigée par Saussure. En effet, il s'agit plutôt d'une reconstitution « publié[e] de manière posthume en 1916 sur la base de notes prises par (…) Charles Bally et Albert Sechehaye » (*ibid.*). Par conséquent, nous ne pouvons pas prétendre être en présence d'une pensée authentiquement saussurienne, pour reprendre les constatations de Loïc Depecker, car « les rédacteurs du Cours n'avaient pas assisté à ses cours de linguistique générale (...) c'est dire si (...) les risques de déformations, gauchissements, contradictions sont nombreux » (Depecker 2012, 3). Plus encore, Simon Bouquet va établir une distinction entre le contenu du CLG et la pensée saussurienne, indiquant que « le *Cours de linguistique générale* a fait subir de nombreuses et sévères distorsions à la pensée de Ferdinand de Saussure, au point qu'il serait plus juste (...) de parler d'un Pseudo-Saussure » (Bouquet 2012, 21). Il convient d'ajouter que Bouquet désigne également les différents manuscrits authentiques de Saussure, publiés bien après la parution du CLG, comme une « linguistique néosaussurienne » (*ibid.*, 22). Des manuscrits qui seront néanmoins publiés bien plus tard que le CLG, toujours d'après les constatations de Bouquet, notamment le manuscrit majeur de « 1957 [où] paraît l'étude du genevois Robert Godel – *Les sources manuscrites du* Cours de linguistique générale » (Bouquet 2004). Toutefois, nous utiliserons exclusivement le CLG dans notre recherche, autrement dit des concepts relevant du Pseudo-Saussure et non pas de la linguistique néosaussurienne. Cela s'explique par le fait que nous considérons le CLG comme étant relativement établi et défini, tandis que la linguistique néosaussurienne relève plutôt de débats qui ne sont pas pertinents dans cette étude.

Néanmoins, si ce CLG est centré sur la linguistique générale, celui-ci considérait hypothétiquement que la sémiologie était une science globale, dont la linguistique ne constituait qu'une composante parmi d'autres. Cependant, le sémiologue Roland Barthes inversa cette conception dite saussurienne, en indiquant que « la linguistique n'est pas une partie (...) de la science générale des

signes, c'est la sémiologie qui est une partie de la linguistique » (Barthes 1964b, 2). De surcroît, le même CLG semblait également donner naissance à un courant théorique majeur, d'après François Dosse, le « structuralisme (…) [qui] trouve son noyau dur, son socle unificateur dans le modèle de la linguistique moderne et dans la figure de (…) son initiateur : Ferdinand de Saussure » (Dosse 2012, ch. 7). Pour autant, Marie-Anne Paveau et Georges-Elia Sarfati soulignent que cet historique du structuralisme « n'est peut-être pas tout à fait rigoureux sur le plan scientifique [puisque] il ne parle pas de "structure" par exemple, mais de "système", [et] le terme *structuralisme* n'apparaissant que vers 1928 » (Paveau et Sarfati 2003, 61). L'influence de la perspective saussurienne est néanmoins majeure, selon Rastier, puisque « la sémiotique contemporaine a été refondée par (...) Saussure, Hjelmslev, Jakobson, Coseriu, Greimas, sans lesquels elle serait restée une branche de la logique » (Rastier 2010, 17).

Une même logique qu'il est possible de situer du côté outre-Atlantique, avec l'émergence de la sémiotique américaine ; celle-ci fut théorisée par Charles Sanders Peirce qui, d'après Christiane Chauviré et « dès les années 1860 (...) conçoit le vaste projet d'une sémiotique ou théorie générale du signe » (Chauviré 1995, 10). Bien ancrée en philosophie, comme exposé précédemment, Jacques Fontanille et Alessandro Zinna indiquent que « la sémiotique inspirée de C. S. Peirce (…) se définit fondamentalement (…) comme une "phanéroscopie" – l'étude des "phanérons", i.e., "ce qui se présente à l'esprit" » (Fontanille et Zinna 2019, 7). Néanmoins, de manière similaire à Saussure, Robert Marty indique que le « corps de doctrine identifié sous le nom de "sémiotique peircienne" (...), [dans les] écrits de Peirce les plus largement diffusés (...) n'existe pas vraiment » (Marty 1990, IX). Pour ainsi dire, « hormis quelques articles formant une unité, les écrits de Peirce restent fragmentaires et inachevés, malgré bien des projets presque tous avortés » (Tiercelin 2013, Introduction), rendant la pensée de Peirce difficilement unifiable. D'après Marty, on peut illustrer ce phénomène par les « 76 définitions du signe relevées dans les manuscrits » (Marty 1990, XI) de Peirce, où les interprétation de son œuvre peuvent donc être multiples et riches. C'est

pourquoi Chandler indique que « Peirce est infiniment ré-interprétable » (Chandler 2022, 38 TDR)[175].

Au-delà de la double origine de cette discipline, nous pouvons souligner que la sémiologie a progressivement gagné en reconnaissance dans le milieu universitaire, mais seulement après quelques décennies. Un constat établi par Klinkenberg, indiquant que cette « discipline (...) n'a fait que récemment son entrée parmi celles qui sont reconnues dans le champ des sciences humaines » (Klinkenberg 2000, ch. 1). Plus encore, il indique surtout que c'est « à partir de la décennie 1960 seulement qu'elle tend à s'institutionnaliser » (*ibid.*). Une institutionnalisation qui passe, selon Rastier, par la « formation de l'Association internationale de sémiotique, en 1969 » (Rastier 2017, 5). Simultanément, l'expansion de cette discipline s'est également déroulée dans le milieu professionnel, selon Karine Berthelot-Guiet, puisque « l'analyse sémiologique appliquée à la communication commerciale (...) s'est ouvert dès la deuxième moitié des années 1960 » (Berthelot-Guiet 2004, 4). Naturellement, c'est pour cette raison que nous estimons que la sémiologie possède *de facto* des perspectives à la fois académiques, ou théoriques, et professionnelles, ou appliquées. Ces dernières permettent *in fine* d'appliquer la sémiologie à n'importe quel domaine, et plus encore pour analyser le développement de jeux vidéo 3D avec des considérations professionnelles.

Cela dit, en ce qui concerne la distinction entre les deux courants de sémiologie européenne et sémiotique américaine évoquée plus haut, on peut souligner qu'elle n'est plus aussi pertinente à l'heure actuelle. C'est pourquoi il existe des ouvrages ou articles scientifiques qui semblent effacer ou ignorer les multiples distinctions historiques pour présenter une discipline *artificiellement* unifiée. Dans de telles situations, et d'après les constatations de Berthelot-Guiet et Jean-Jacques Boutaud, les termes *sémiologie* et *sémiotique* sont souvent utilisés de manière interchangeable, puisque « l'usage retient indifféremment les deux étiquettes ou bannières (...) [et] le débat est ailleurs » (Berthelot-Guiet et Boutaud 2015, 5). Naturellement, un tel procédé ne peut aucunement éluder le fait qu'il existe, comme rappelé par Winfried Nöth, « de nombreuses écoles et

175. « Peirce is endlessly reinterpretable » (Chandler 2022, 38).

courants aussi bien théoriques qu'appliqués de la sémiotique » (Nöth 1990, 3 TDR)[176].

En revanche, au regard de la rigueur que nous souhaitons imposer à notre recherche, il nous semble important de spécifier les acceptions des termes *sémiologie* et *sémiotique*. Cette distinction que nous allons élaborer n'est nullement d'ordre géographique, ce qui signifie que nous ignorerons la provenance européenne ou américaine des courants, et cela tout au long de l'étude. Ainsi, on adoptera plutôt une signification conventionnelle, illustrée par une dualité entre des perspectives théoriques et appliquées que nous venons d'exposer, ce qui est bien mis en exergue par Klinkenberg, où « *sémiologie* désigne (…) la discipline qui couvre tous les types de langage, [et] *sémiotique* désignant un des objets dont peut s'occuper cette discipline » (Klinkenberg 2000, ch. 1). Au regard de nos multiples objectifs, il nous semble que cette distinction est claire, concise et compréhensible par l'ensemble des potentiels lecteurs, y compris par d'éventuels néophytes en sémiologie. Une même distinction primaire que nous allons définir, où la sémiologie théorique désigne une recherche purement théorique, et la sémiotique appliquée une étude portant sur un domaine particulier. Elle signifie ainsi que la sémiotique appliquée porte sur des objets précis, pour reprendre Klinkenberg, puisque « la langue est une sémiotique, comme le sont aussi les pictogrammes (...) [ou] le mobilier, (...) chacune de ces sémiotiques est donc une actualisation de la sémiologie, discipline générale » (Klinkenberg 2000, ch. 1).

Cette proposition fait référence à des distinctions similaires, déjà mises en exergue par différents sémioticiens ; on peut notamment lister, grâce à Louis Hébert, « Morris [qui] a proposé (...) la sémiotique *pure*, *descriptive*, et *appliquée* (...), Carnap a ajouté (...) la sémiotique *générale* et *spéciale* (...), et Hjelmslev a fait une distinction entre *sémiologie* (...) et *méta-sémiologie* » (Hébert 2020, 5 TDR) [177] . Néanmoins, nous souhaitons formuler une opposition limpide et relativement simple, compréhensible par des néophytes dans ce domaine.

176. « many schools and branches of both theoretical and applied semiotics. » (Nöth 1990, 3).
177. « Morris [which] proposed (...) *pure*, *descriptive*, and *applied* semiotics (...), Carnap added (...) *general* and *special* semiotics (...), and Hjelmslev distinguished between *semiology* (...) and *metasemiology* » (Hébert 2020, 5).

C'est pourquoi nous considérons qu'une analyse appartient à la sémiologie théorique si elle porte uniquement sur des aspects exclusivement théoriques et conceptuels. Par exemple, celle-ci pourrait questionner l'impact des concepts du sémioticien Yuri Lotman en sémiologie, et sa relation avec d'autres concepts, écoles de pensée ou courants. L'objet d'une telle recherche serait *in fine* la sémiologie elle-même, ou plutôt *en elle-même et pour elle-même*, pour reprendre mot pour mot la formulation de « la phrase apocryphe finale (...) de F. Bopp (..) et non de F. de Saussure » (Rastier 2012, 10). Et l'objectif final serait de concevoir de nouvelles théories exclusivement pour la sémiologie, où des exemples pratiques ne serviraient que d'illustration. Néanmoins, sa particularité réside principalement dans sa capacité à approfondir substantiellement les concepts et définitions sémiologiques, à les contextualiser mais également à mettre en exergue les différents courants de pensée. Elle s'adresse donc *in fine* exclusivement à des sémioticiens ou des sémiologues.

En revanche, une analyse de sémiotique appliquée porterait plutôt sur une étude qui exploite les multiples concepts sémiotiques, par exemple le signe linguistique saussurien, dans des domaines comme le cinéma ou, dans notre cas, l'informatique. L'intérêt majeur serait de pouvoir utiliser ces multiples outils conceptuels pour élaborer des théories novatrices, à travers un prisme rigoureusement scientifique, et également répondre à des besoins industriels. Dans notre thèse, celle-ci répondrait en l'occurrence aux besoins de l'industrie du développement de jeux vidéo 3D. C'est pourquoi elle s'adresse exclusivement à des professionnels d'un secteur précis, par exemple à des professionnels de l'industrie cinématographique ou vidéoludique, et non pas à des sémiologues ou sémioticiens. Elle pourrait donc s'intégrer harmonieusement dans des pôles recherche et développement d'entreprises privées, et utiliserait la méthodologie scientifique, de surcroît universitaire, pour faire progresser l'industrie entière. Néanmoins, une telle étude ne pourrait aucunement s'immiscer dans d'éventuels débats sémiologiques. Elle ne cherchera pas à lister les multiples définitions d'un même concept, ou à retracer l'historique, voire la contextualisation d'une théorie.

Concrètement, cela signifie également que notre thèse doctorale s'inscrit certes dans la discipline générale de l'étude du signe, la *sémiologie*, mais elle applique celle-ci à un objet particulier, une *sémiotique* qui traite du développement de jeux vidéo en 3D temps réel. En d'autres termes, il s'agit d'une

étude de sémiotique appliquée, s'adressant exclusivement à l'industrie vidéoludique et dont l'objectif principal est d'améliorer le cadre théorique ou conceptuel. Précisons que notre étude a nécessité le développement de notre approche novatrice du Développement Logiciel Tridimensionnel.

Par ailleurs, il est important de revenir sur un point précis concernant les sémiotiques appliquées, puisque, comme rappelé par Alain Eraly, « si le champ de la sémiotique semble a *priori* plus cohérent, il suscite depuis toujours des controverses qui portent sur ses concepts les plus fondamentaux » (Eraly 2011, 167). Or, ces dissensions ne présentent généralement aucun intérêt pour une problématique de sémiotique appliquée, y compris pour la nôtre, centrée sur les spécificités du développement de jeux vidéo. En substance, et pour prolonger notre propos, cela signifie que nous ignorerons les débats historiques, sémantiques ou géographiques de la discipline, et cela tout au long de notre thèse. Les recontextualisations de la création originelle des multiples concepts, théories et outils sémiologiques seront au mieux parcellaires, sinon absentes, sauf si elles permettent de mieux percevoir un élément de notre problématique. Surtout, une recontextualisation ne répondrait aucunement aux objectifs d'une recherche universitaire en sémiotique appliquée.

Plus encore, nous prenons la liberté de pouvoir juxtaposer des concepts sémiologiques pouvant appartenir à des auteurs ou des écoles potentiellement opposées ou paradoxales, si cela est nécessaire pour formuler des arguments pertinents face à notre problématique. Un point fondamental à prendre en considération car celui-ci va influencer la structuration de notre recherche. Désormais, et au regard de l'ensemble des éléments présentés, nous sommes justement en mesure de révéler cette structuration. Ainsi, nous pouvons désormais exposer le plan complet de notre étude doctorale, et ensuite passer au deuxième chapitre pour plonger au cœur de notre approche sémio-informatique, le Développement Logiciel Tridimensionnel.

1-3. STRUCTURE GLOBALE DE LA THÈSE

Notre recherche doctorale est structurée en sept parties distinctes : une importante introduction, cinq chapitres majeurs ainsi qu'une conclusion finale. Chaque chapitre est soigneusement élaboré pour esquisser de multiples pistes en rapport avec notre problématique, tout en favorisant une progression linéaire de notre discours. Illustrons cet aspect par le quatrième chapitre, qui perfectionnera et affinera les théories élaborées dans les chapitres précédents, tout en introduisant de nouvelles théories. Au regard de l'organisation globale de la thèse, tous les chapitres, en incluant également l'introduction et la conclusion, possèdent une même structure triptyque : deux sections principales de développement ainsi qu'une troisième section synthétisant l'ensemble du chapitre. Plus encore, chaque section principale est également fragmentée en une structure tripartite, comprenant trois sous-sections.

Spécifions également que l'étude diachronique sur les multiples générations de consoles débutera dès le troisième chapitre, et cela jusqu'au sixième inclus. Chaque chapitre sera ainsi consacré à une génération de consoles particulière, respectant un ordre chronologique. À l'intérieur de chaque chapitre, les deux sections principales seront systématiquement structurées de la manière suivante : la première section sera consacrée à l'historique du développement de jeux sous l'angle de la programmation ; la deuxième s'attellera à développer et approfondir des théories de sémiotique appliquée, à l'aune des faits historiques et techniques de la section précédente. Enfin, la troisième section synthétisera simplement l'ensemble des résultats et théories du chapitre concerné. Sur ce dernier point, nous avons conscience que cela implique de réitérer à plusieurs reprises nos propos. Néanmoins, nous estimons cela nécessaire, à la fois pour désambiguïser certaines parties de notre recherche, mais également pour faciliter la compréhension des lecteurs, puisque cette recherche interdisciplinaire intègre sémiotique et informatique, deux disciplines relativement distinctes, voire éloignées. L'accessibilité aux théories et à une compréhension limpide de notre propos était donc une priorité, un aspect capital dans notre étude. Désormais, nous allons résumer l'introduction et ensuite détailler le contenu des chapitres majeurs de notre thèse.

1. INTRODUCTION

Tout d'abord, si cette introduction est tout aussi exhaustive que les différents chapitres ultérieurs, c'est parce qu'il était nécessaire de poser les jalons d'une problématique particulièrement complexe et fondamentalement ancrée dans deux disciplines majeures et éminemment riches : l'informatique et la sémiologie.

Ainsi, la première section majeure, nommée **Contextualisation multifacette**, a exposé le contexte global de la thèse, la taxinomie des genres vidéoludiques, la problématique ainsi que les deux corpus. **L'introduction** de celle-ci a donc présenté le contexte justifiant notre recherche, c'est-à-dire l'observation de la combinaison des genres vidéoludiques dans le *gameplay* des jeux en monde ouvert. Nous avons ensuite succinctement présenté les objectifs majeurs de la thèse, à savoir la création d'une structure sémio-informatique du développement de jeux vidéo en 3D temps réel via notre approche sémio-informatique. Puis nous avons mis en relief les prémices de l'histoire des jeux vidéo, débutant avec les premiers concepts des années 1950, notamment le *Cathode Ray Tube Amusement Device* (Djaouti 2019). Nous avons terminé cet historique à la fin de la deuxième génération de consoles, avec le krach vidéoludique de 1983 causé par plusieurs facteurs, notamment une saturation du marché et une faible qualité des jeux (Ichbiah 2014 ; Kent 2001 ; Ernkvist 2008). L'objectif principal était de mieux comprendre le contexte d'apparition de la typologie des genres vidéoludiques. Néanmoins, aborder cet historique vidéoludique en amont nous semblait essentiel pour appréhender les enjeux de notre thèse ainsi que notre problématique. Ceci ne permet pas, pour autant, de pouvoir répondre à notre problématique, puisqu'il ne s'agit que du début de l'histoire des jeux vidéo, alors que cette problématique est plutôt centrée sur les générations ultérieures.

Ensuite, la première sous-section intitulée **Taxinomie des genres vidéoludiques** a permis de présenter la potentielle première taxinomie théorique des genres vidéoludiques (Crawford 1984). Nous avons également analysé la troisième génération de consoles, notamment la *Sega SG-1000* ainsi que la *NES*. Cette dernière console a permis d'introduire le *Nintendo Seal of Quality* (Scullion 2019 ; Levy et Novak 2010) ainsi qu'un classement des jeux par genre (Doucet

2020). L'objectif était hypothétiquement de prémunir le marché d'un nouveau krach similaire à celui de 1983 (Levy et Novak 2010 ; Ichbiah 2014).

Puis, dès la deuxième sous-section, nommée **Problématique polymorphique**, nous avons exposé la problématique centrale de notre thèse ainsi que l'hypothèse de départ hypothético-déductive, qui conditionne également notre démarche scientifique.

Enfin, la troisième sous-section, **Corpus développemental et vidéoludique**, a permis de présenter deux catégories de corpus pour une approche qualitative : le corpus développemental, subdivisé en corpus développemental direct et corpus développemental indirect, ainsi que le corpus vidéoludique. Nous pouvons présenter ces multiples éléments dans le tableau ci-dessous.

Problématique	À l'aune des moteurs de jeu des années 2015, comment la création des mécanismes interactifs dans le développement de jeux vidéo en 3D temps réel et depuis les années 1990 est-elle symbolisée, structurée, élaborée et limitée par les genres vidéoludiques ?
Hypothèse	Les genres vidéoludiques reflètent des conventions entièrement arbitraires provenant de développeurs, joueurs, jeux vidéo commercialisés et journalistes. Ces conventions ne résultent aucunement de limitations technologiques du développement de jeux vidéo en 3D temps réel dans les années 2015.
Corpus	Corpus développemental direct : Blueprints, C++ 14, Unreal Engine 4.27. Corpus développemental indirect : cf. *Annexe D.* Corpus vidéoludique : cf. *Annexe B*.

Tableau 10. Problématique, hypothèse et corpus de la recherche doctorale

Au-delà de ces aspects primordiaux, la deuxième section majeure, **Scientifisation des jeux vidéo**, a permis de définir d'abord la complexité inhérente à l'étude scientifique du jeu, puis les problèmes liés au champ des *Game Studies*, et enfin les deux domaines cohabitant dans cette recherche, à savoir l'informatique et la sémiologie.

En **introduction** de celle-ci, nous avons exposé les difficultés inhérentes à l'étude scientifique du jeu vidéo (Juul 2000 ; McShaffry et Graham 2013 ; Wang et Nordmark 2015) et le processus de scientifisation des jeux dans l'espace universitaire. Puis nous avons constaté une filiation des *Game Studies*, champ

disciplinaire étudiant les jeux vidéo, au domaine général du jeu et du ludique (Leconte 2019 ; Bonenfant 2010 ; Di Filippo 2014). Ce phénomène de filiation occulte potentiellement toutes les spécificités du jeu vidéo en tant que logiciel informatique, puisque l'aspect technique semble inexistant dans les *Game Studies*. On trouve également de nombreuses études vidéoludiques portant sur les comportements engendrés par l'activité des jeux vidéo, telles que le lien entre comportements violents et pratiques vidéoludiques. Néanmoins, certaines méta-études ne permettent aucunement de pouvoir définitivement répondre à cette sempiternelle question (Drummond, Sauer et Ferguson 2020). Les résultats ne sont pas unifiés pour autant (Hodent 2021), puisque d'autres méta-études considèrent une potentielle augmentation dite légère de la violence (Mathur et VanderWeele 2019). C'est pourquoi il nous semble nécessaire que les recherches s'accentuent et se poursuivent pour traiter, au mieux, de l'ensemble de ces questions d'une importance capitale (Ferguson 2015), en plus d'autres questions portant, par exemple, sur l'addiction (Limone, Ragni et Toto 2023).

Puis la première sous-section, **Incompatibilité des Game Studies**, a permis d'analyser les limitations des *Game Studies*, à travers les cinq types d'analyses et méthodologies exposées par les travaux d'Egenfeldt-Nielsen, Smith et Tosca (2015) : le jeu, le joueur, la culture, l'ontologie et le métrique. Grâce à ces types d'analyses exposés, nous pouvons justifier une incompatibilité entre notre problématique et le cadre théorique du champ des *Game Studies*. Ce dernier semble, en effet, naturellement peu enclin à être rigoureusement exploitable dans le cadre d'analyses techniques ou informatiques, et les cinq types d'analyses proposés ne peuvent aucunement permettre de répondre à notre problématique. Cet aspect est la raison principale ayant engendré la création de notre nouvelle approche sémio-informatique, où notre objectif est surtout d'imposer une rigueur et méthodologie scientifique répondant à des besoins vidéoludiques industriels.

Nous avons ensuite utilisé la deuxième sous-section **Informatique et développement de jeux** pour exposer les aspects technologiques principaux que nous allons étudier, c'est-à-dire des langages de programmation ainsi que de multiples moteurs de jeu. Puis nous avons présenté deux typologies de langages de programmation : une première en cinq générations de langages (Dixit 2011), et une seconde en plusieurs niveaux de langages (Hugg 2018) que nous utiliserons tout au long de notre thèse. En outre, nous avons également précisé

que les termes techniques en anglais seront majoritairement utilisés dans notre étude, sauf s'il existe une traduction française attestée, puisque les ouvrages, documentations et logiciels sont exclusivement en anglais (Guo 2018 ; Saint 2013).

Enfin, la troisième sous-section **Sémiologie et sémiotiques** a d'abord présenté la sémiologie, une science étudiant la représentation de la réalité à travers des systèmes de signes et processus de signification (Chandler 2022). L'étude de cette science débute par l'histoire de la double origine du domaine, c'est-à-dire la sémiologie Européenne de Saussure, liée à la linguistique et la sémiotique d'outre-Atlantique de Peirce, ancrée dans la logique (Rastier 2001 ; Hénault 2008 ; Badir 2010). Nous présentons ensuite une distinction entre deux approches mises en exergue par plusieurs auteurs (Klinkenberg, Morris, Carnap, Hjelmslev) et ayant des implications importantes pour une étude scientifique : la sémiologie théorique et la sémiotique appliquée. La première étudie les théories et concepts de la sémiologie pour les analyser, et s'adresse exclusivement à des sémioticiens ou sémiologues. La seconde, quant à elle, relève plutôt d'une volonté d'appliquer des concepts et théories à un domaine précis, par exemple le cinéma ou le développement de jeux vidéo. Elle répond donc à des besoins industriels, s'intégrant harmonieusement dans des pôles recherche et développement d'entreprises privées et s'adressant *in fine* à des professionnels. C'est justement cette seconde approche que nous utilisons dans notre recherche, c'est-à-dire une sémiotique appliquée répondant à des besoins pratiques de l'industrie vidéoludique, avec l'intégration d'une démarche scientifique pour développer des théories et structures sémiotiques.

2. Développement Logiciel Tridimensionnel

Le deuxième chapitre de notre thèse doctorale va conceptualiser rigoureusement les différents fondements scientifiques et théoriques de cette recherche. Surtout, celui-ci définira les caractéristiques de notre approche sémio-informatique inédite, nommée Développement Logiciel Tridimensionnel (désormais DLT). Une approche qui permettra surtout de dépasser les multiples limitations des *Game Studies*, en replaçant les outils de développement de jeux vidéo 3D et une haute technicité au cœur de la démarche scientifique. De surcroît, celle-ci est composée de deux principes immuables : la passerelle technologique

infrangible et le générateur de théories vidéoludiques. Le premier principe sera un support pour consolider le lien avec l'industrie vidéoludique, alors que le second présentera des fondements théoriques pour concevoir des théories et structures sémio-informatiques *ex nihilo* ou *ex materia*.

3. CINQUIÈME GÉNÉRATION

Ce troisième chapitre va d'abord évoquer succinctement la quatrième génération de consoles, avec une mise en exergue du développement de jeux sur *Super Nintendo* et l'utilisation de la famille des langages *assembleurs*. Ensuite, nous présenterons le développement de jeux pendant la cinquième génération, débutant au milieu des années 1990 avec la *Playstation*, la *Nintendo 64* et la *Sega Saturn*. On introduira le langage de programmation *C*, la standardisation de multiples API comme *OpenGL* et *Direct3D* puis le contexte de développement des premiers moteurs de jeu, et particulièrement *Unreal Engine*. On étudiera également les possibilités du développement amateur de jeux vidéo avec la *Net Yaroze* et le *Game Basic for Sega Saturn*. Ces données historiques serviront de base pour élaborer des concepts sémio-informatiques, en débutant par celui du signe interactif, qui s'appuie sur le signe linguistique saussurien et la subdivision de Hjelmslev. On reprendra également des concepts basés sur la sémiotique tensive de Zilberberg, des notions du schéma actantiel pour mieux appréhender l'interactivité, puis nous évoquerons les premiers mondes ouverts en trois dimensions.

4. SIXIÈME GÉNÉRATION

Au regard du quatrième chapitre, celui-ci est consacré à la sixième génération de consoles, débutant dès 1998 avec la *Dreamcast*, la *PlayStation 2*, la *GameCube* et enfin la *Xbox*. Nous évoquerons d'abord la démocratisation du langage *C++* et de son paradigme orienté objet dans le développement de jeux vidéo. On illustrera ensuite la généralisation des premiers moteurs et outils de développement commerciaux, en particulier *Unreal Engine 2* et les deux versions de *RenderWare*. Sur le plan du développement amateur de jeux, on analysera le *Linux Kit* de la PS2, le langage *YA-Basic* mais également l'outil *DarkBasic Pro*, tout en mentionnant trois autres moteurs, *Ogre, Crystal Space* et *Irrlicht*. Cet aspect historique permettra de développer des concepts sémio-informatiques,

en commençant par spécifier l'arbitraire et la linéarité des signes interactifs, puis en explorant leur mutabilité et immutabilité. On développera ensuite la théorie de l'interactivité générationnelle, puis le paradoxe de l'interactivité manichéenne et enfin la théorie des faisceaux et substrats des univers virtuels.

5. SEPTIÈME GÉNÉRATION

Ce cinquième chapitre va esquisser les contours de la septième génération de consoles, ayant commencé dès 2005 avec la *Xbox 360*, puis la *PlayStation 3* et la *Wii*. On observera une tendance à la généralisation du développement de moteurs à la fois internes, mais également sous licence. De surcroît, la sphère publique va témoigner de l'apparition de deux moteurs majeurs : *Unreal Development Kit*, version publique du moteur *Unreal Engine 3*, et *Unity*. Cette situation nouvelle dans l'industrie permettra de développer différents concepts sémio-informatiques, notamment une précision technologique sur les interactants mais également une vision développementale du monde virtuel, basée sur l'hypothèse Sapir-Whorf. On pourra également élaborer une analyse plus fine des signes interactifs, notamment avec notre concept du niveau sémique. Enfin, nous terminerons ce chapitre par une analyse plus approfondie des mondes ouverts, grâce à l'élaboration du concept d'isotopie interactive.

6. HUITIÈME GÉNÉRATION

Ce sixième et dernier chapitre de la partie développement de notre thèse doctorale est d'abord consacré à la huitième génération de consoles, ayant démarrée dès 2012 et marquée par l'arrivée des consoles *PlayStation 4*, *Xbox One*, *Wii U* et *Switch*. On observera, dans la sphère publique, une expansion importante du moteur *Unity*, mais également d'autres moteurs comme *Lumberyard* ou encore *Stingray*. Nous analyserons également la démocratisation publique du moteur *Unreal Engine 4,* et les conséquences pour l'industrie vidéoludique. Cela permettra ensuite de clôturer l'analyse diachronique pour élaborer notre hyperstructure 3D, qui va permettre d'unifier l'ensemble des théories développées dans les chapitres précédents.

7. CONCLUSION

Nous terminerons notre thèse doctorale par une conventionnelle conclusion, qui synthétisera la totalité de notre recherche et présentera les résultats, une rétrospection, les apports scientifiques et industriels, et enfin quelques perspectives pouvant mettre en lumière des sujets connexes.

2. DÉVELOPPEMENT LOGICIEL TRIDIMENSIONNEL : SPÉCIFICATIONS

2-1. PASSERELLE TECHNOLOGIQUE INFRANGIBLE

À la lumière du contexte protéiforme exposé en introduction, et plus particulièrement avec la mise en surbrillance des limitations des *Game Studies*, nous avons élaboré une nouvelle approche sémio-informatique, innovante et rigoureusement structurée. Nommée Développement Logiciel Tridimensionnel (désormais DLT), son objectif majeur est de créer un cadre théorique sémiotique solide permettant de mieux analyser, appréhender et comprendre les outils informatiques utilisés dans le développement de jeux vidéo en 3D temps réel. Surtout, son avantage substantiel est de réintégrer une haute technicité au cœur des analyses, s'arc-boutant sur des technologies informatiques complexes et protéiformes.

De surcroît, nous avons développé et peaufiné l'élaboration de cette approche en tenant compte des multiples analyses et résultats de cette recherche, et cela jusqu'à la finalisation de notre manuscrit de thèse. Certes, présenter celle-ci dès le deuxième chapitre pourrait sembler paradoxal, mais nous estimons que cela permet de garantir la limpidité de nos chapitres ultérieurs. En effet, certains concepts de notre approche sont transversaux, et ceux-ci sont surtout utilisés dans la majeure partie des chapitres. C'est pourquoi nous préférons éviter de parsemer nos différents chapitres d'explications répétitives sur cette approche, en concentrant le tout dans ce deuxième chapitre. Par ailleurs, nous avons d'emblée conçu l'approche du DLT en lui intégrant deux principes immuables et inédits : une passerelle technologique infrangible ainsi qu'un générateur de théories vidéoludiques. Nous allons d'abord présenter succinctement ces deux principes éminemment importants dans cette introduction, pour ensuite les exposer longuement dans les deux sous-sections principales de ce chapitre.

Tout d'abord, ce que nous entendons par notre formulation inédite de passerelle technologique infrangible, c'est un principe immuable qui sous-tend notre recherche entière. Celui-ci permet de mettre en surbrillance l'un des objectifs majeurs de notre thèse de sémiotique appliquée : créer une *passerelle* entre des analyses scientifiques, ou sémiotiques dans ce cas, portant sur la création de l'interactivité et des considérations technologiques, ou informatiques, de l'industrie du jeu vidéo, principalement à travers des outils professionnels.

Ensuite, si nous utilisons le sens figuré de *passerelle*, de surcroît *infrangible*, c'est bien pour mettre en relief ce même objectif de maintenir continuellement ce lien avec l'industrie vidéoludique. Il s'agit donc de pouvoir comprendre les multiples facettes du développement de jeux vidéo, d'un point de vue exclusivement technique, en se concentrant sur les outils de conception. Et, grâce à cette même passerelle, nous pourrons ensuite élaborer plusieurs théories qui seront *in fine* ancrées dans des considérations à la fois technologiques et industrielles. C'est pourquoi cette création de théories est le rôle du second principe immuable de notre approche sémio-informatique, le générateur de théories vidéoludiques. Précisons également que, comme le souligne l'adjectif *technologique*, nous ne questionnerons pas les aspects organisationnels du développement. Nous nous concentrerons exclusivement sur les possibilités émanant des outils, et les questions sous-jacentes portant sur les méthodes de gestion de développement seront éludées, car trop éloignées de notre problématique.

En revanche, nous précisons que l'orientation éminemment technologique de notre recherche permet surtout de contraster, de manière plus importante encore, avec la définition habituellement plurielle et protéiforme des jeux vidéo. Surtout, nous considérons les jeux vidéo comme étant des logiciels informatiques, puisque nous analysons exclusivement l'aspect technique de ceux-ci. Nous pouvons également nous appuyer sur la définition de Sergei Savchenko, indiquant que les « graphismes 3D interactifs suggèrent (...) deux composantes (...) : *les graphismes* (représentant des images sur un dispositif d'affichage) et *l'interactivité* (réagissant aux *inputs* d'un utilisateur lors de l'exécution d'une application) » (Savchenko 2000, 12 TDR)[178]. C'est pourquoi nous ignorons d'autres aspects, par exemple exposés par Galanina, Akchelov et Sakharova, c'est-à-dire le fait qu'un « jeu vidéo est aussi bien un code ou une plate-forme qu'un discours, un media digital, une nouvelle forme artistique, etc. » (Galanina, Akchelov et Sakharova 2020, 39 TDR)[179]. Néanmoins, cette définition

178. « interactive 3D graphics suggests (...) two components (...): *graphics* (representing images on a display device) and *interactive* (reacting to a user's input in the course of an application's execution) » (Savchenko 2000, 12).

179. « Among other, (...) video game is code, a platform, as well as a narrative, digital media, a new form of art, etc. » (Galanina, Akchelov et Sakharova 2020, 39).

rédigée par Galanina *et al.*, recouvrant un spectre bien plus large, n'est nullement fallacieuse ou erronée. Nous estimons, au contraire, qu'elle pourrait correspondre à d'autres recherches ou discours différents sur le jeu vidéo, notamment ceux des *Game Studies,* et particulièrement avec les types d'analyses évoqués en introduction. De plus, à travers le prisme informatique des jeux vidéo utilisé dans cette recherche, nous pouvons les distinguer des autres logiciels grâce à, selon Rouse III, une « composante (...) qui n'est trouvable dans aucune autre forme d'art : l'interactivité » (Rouse III 2005, XX TDR)[180]. Ce qui signifie concrètement que notre définition reflète les intérêts de notre recherche ainsi que de notre problématique, sans pour autant exclure celle de Galanina *et al*.

Au-delà de l'interactivité, les caractéristiques du *développement* d'un jeu vidéo sont également substantiellement différentes de celles d'autres types de logiciels informatiques. C'est pourquoi on peut considérer, comme rappelé par Ian Millington, que « le développement de jeux vidéo vit dans son propre monde technique (...) [avec] son propre langage, ses compétences et ses défis » (Millington 2019, 3 TDR)[181]. Les outils de création de jeux vidéo sembleraient élaborés sur mesure, pour répondre à des besoins uniques, plutôt que des outils généralistes. Et cela malgré le constat exposé par Millington de « nombreux efforts pour standardiser le développement de jeux vidéo, de le conformer avec les autres industries de l'informatique » (*ibid*. TDR)[182], phénomène que nous observerons au fil des générations.

Par ailleurs, en plus de considérer le DLT exclusif aux jeux vidéo, nous pourrions aller plus loin dans sa caractérisation, en indiquant que cette approche a été élaborée uniquement en tenant compte de *notre* problématique ainsi que *notre* contexte de recherche. Elle n'a donc nullement vocation à être démocratisée, et encore moins à devenir une approche générique et applicable à n'importe quel type de logiciel. Par exemple, elle n'est potentiellement pas pertinente pour traiter une problématique portant sur la conception d'une

180. « component (..) that is found in no other art form: interactivity » (Rouse III 2005, XX).
181. « game development lives in its own technical world (...) [with] its own idioms, skills, and challenges » (Millington 2019, 3).
182. « numerous efforts to standardize game development, in line with the rest of the software industry » (Millington 2019, 3).

application web ou d'un système d'exploitation. Elle représente plutôt un hapax conçu pour traiter notre problématique avec la plus grande précision possible, en vue d'obtenir des résultats probants. Surtout, la conception de cette approche *ex nihilo* a été réalisée par pure nécessité au regard de la situation actuelle de l'étude du développement de jeux vidéo. Il ne s'agit donc aucunement d'un objectif initial de notre thèse, mais seulement d'une conséquence induite par la singularité de notre problématique. À ce titre, nous ne souhaitions pas intégrer dans notre étude une discussion épistémologique portant sur les études des jeux vidéo ou logiciels informatiques, que nous considérions comme étant superfétatoires au regard de cette thèse relativement conséquente. De surcroît, l'aspect technique nous semble profondément complexe pour, en plus, l'agrémenter d'une argumentation sur des principes scientifiques. Mais, au regard des multiples difficultés rencontrées, il a été nécessaire, au fur et à mesure de notre recherche, d'élaborer l'approche du DLT, puisque la démarche scientifique et universitaire requiert une importante rigueur.

C'est pourquoi nous pouvons désormais élaborer une méthodologie scientifique pour notre approche sémio-informatique du DLT. Celle-ci permettra d'orienter notre recherche afin qu'elle puisse répondre aux besoins conceptuels et théoriques de l'industrie vidéoludique. Notre méthodologie est composée de plusieurs étapes obligatoires, que nous listons dans le tableau ci-dessous, pour les redéfinir plus bas, mais également renforcer la pertinence de celles-ci dans l'ensemble de ce chapitre.

Étape	Description
Problématisation	Définition de la problématique servant de fil directeur
Hypothèse	Élaboration d'une hypothèse hypothético-déductive
Corpus	Constitution des éléments du corpus développemental et corpus vidéoludique
Contextualisation	Revue de la littérature historique de l'industrie, puis scientifique et informatique
Analyse	Analyse des données historico-scientifiques
Théorisation	Élaboration de théories scientifiques

Tableau 11. Étapes du module de scientificité

Ainsi, nous avons déjà traité en introduction des trois premières étapes de la méthodologie de notre DLT, notamment la problématisation, l'élaboration d'une hypothèse ainsi que la constitution des deux corpus. Désormais, nous pouvons planifier une contextualisation historico-scientifique. L'objectif de cette section est de pouvoir réaliser à la fois un historique du développement de jeux vidéo pour une génération donnée, mais également une revue de la littérature scientifique sur le sujet. Une littérature relativement parcellaire pour le développement de jeux vidéo, de surcroît sur les moteurs de jeu, puisqu'il s'agit d'un domaine très peu étudié sur le plan scientifique.

Par la suite, l'analyse et l'exploitation des données permettront de créer des théories scientifiques sur le développement, qui *in fine* répondront à notre problématique. Néanmoins, notre méthodologie scientifique est exclusivement applicable, par exemple, pour un pôle de recherche et développement d'une entreprise. C'est pourquoi la réalisation d'entretiens est *de facto* formellement proscrite dans notre approche, puisqu'une entreprise ne pourrait aucunement réaliser des entretiens avec d'autres entreprises concurrentes, notamment pour des raisons légales.

Désormais, puisque nous avons précisé le contexte de conception mais également les spécificités de notre approche du DLT, nous pouvons détailler le premier principe immuable, la passerelle technologique infrangible. Ce principe est constitué de trois fondements majeurs qui permettent de répondre aux difficultés inhérentes posées par l'étude technique du développement de jeux vidéo 3D :

- La fortification technologique ;
- L'obsolescence inéluctable ;
- La véridiction technico-discursive.

Dans l'ordre présenté, le premier fondement est centré sur une utilisation cohérente du corpus développemental direct au regard de notre problématique sémio-informatique. Puis le deuxième fondement traite de multiples complications qui se présentent lors d'une étude diachronique, permettant *de facto* de mieux utiliser le corpus développemental indirect. Enfin, le troisième fondement évoque le traitement du discours des développeurs portant sur les outils de développement de jeux vidéo. Dès la prochaine section, nous allons nous intéresser au premier fondement de cette passerelle technologique infrangible.

2-1-A. FORTIFICATION TECHNOLOGIQUE

Nommé fortification technologique, ce premier fondement de notre passerelle technologique infrangible implique de mettre en surbrillance plusieurs aspects technologiques du développement de jeux vidéo. Il vise principalement à confirmer la pertinence des outils informatiques pour notre problématique, tout en permettant *in fine* un meilleur usage de notre corpus développemental direct présenté en introduction. Surtout, nous considérons que notre approche se base fondamentalement sur la modularité, et l'idée de modulariser celle-ci s'inspire de l'informatique.

En effet, nous avons constaté que les modules sont aussi bien utilisés dans les langages de programmation comme le *C++* que dans les moteurs de jeu. Par exemple, les modules sont disponibles depuis le C++ 20 (Horton et Van Weert 2020, 365). Et, pour le moteur *Unreal Engine*, les « modules sont les éléments de base d'Unreal Engine (UE) [et] le moteur est implémenté sous forme d'une grande collection de modules, et les jeux fournissent leurs propres modules pour les compléter » (Epic Games 2023, *Modules Properties* TDR)[183]. Dans tous les cas, en nous basant sur l'ensemble de nos recherches, nous proposons ainsi trois modules inédits qui émergent de ce fondement :

- ► Le module de précision technologique ;
- ► Le module de commercialisation ;
- ► Le module de rationalisation.

Tout d'abord, la conception de notre module de précision technologique résulte d'une forte richesse et hétérogénéité des outils de développement de jeux disponibles, comme nous l'observerons tout au long de la recherche. L'objectif est de centrer et structurer nos analyses et recherches sur des technologies précises. De surcroît, on remarque également la possibilité d'étudier un même phénomène à travers plusieurs niveaux techniques ou technologiques différents. C'est pourquoi nous estimons que le niveau d'analyse doit être en concordance et en harmonie avec la problématique de l'étude. Pour rendre notre propos plus

183. « modules are the building blocks of Unreal Engine (UE) [and] the engine is implemented as a large collection of modules, and games supply their own modules to augment them » (Epic Games 2023, *Modules Properties*).

intelligible, nous présentons dans le tableau ci-dessous une liste de différents modules d'analyse plausibles que nous pouvons envisager au regard d'une problématique technique portant sur le domaine vidéoludique. Pour cela, nous nous appuyons sur la pléthore d'outils répertoriés et cités dans notre recherche. Naturellement, ce tableau est schématique et non exhaustif, mais permet de rendre compte de la profonde complexité dans le choix du module d'analyse.

Module d'analyse	Exemples
Langage de programmation	C, C++, C#, Java, Python...
API et API 3D	DirectX, OpenGL, Metal, Vulkan...
Moteur de rendu 3D	OGRE, Irrlicht...
Moteur de jeu	Godot, Unity, Unreal Engine...
Logiciel de créations	RPG Maker...

Tableau 12. Différents modules technologiques du DLT

Ainsi, la détermination du module d'analyse dépend de la proximité entre notre problématique soulevée et les technologies correspondantes. Nous considérons surtout qu'il est nécessaire de choisir un seul et unique module d'analyse, afin de ne pas complexifier inutilement notre recherche. Et, au regard de notre problématique, c'est-à-dire l'étude de l'évolution des genres à travers la conception de jeux, nous estimons qu'il existe un seul module d'analyse cohérent et pertinent : le moteur de jeu. D'une importance capitale pour créer l'interactivité dans le développement de jeux, on peut définir celui-ci, grâce à Morgan McGuire et Odest Chadwicke Jenkins, comme étant un logiciel qui « fournit presque tous les composants et la plupart des outils nécessaires à la création d'un jeu » (McGuire et Jenkins 2008, 128 TDR)[184]. Surtout, Gregory ajoute le critère de réutilisation, puisqu'il indique que « nous devrions probablement réserver le terme "moteur de jeu" pour désigner un logiciel extensible et pouvant servir de base pour de nombreux jeux sans modification majeure » (Gregory 2019, 12

184. « provide almost all the components and most of the tools needed for producing a game » (McGuire et Jenkins 2008, 128).

TDR)[185]. Naturellement, cette définition primaire que nous venons d'esquisser sera profondément complexifiée par la suite, plus précisément dès le prochain chapitre, et nous ne pourrions pas décrire les spécificités d'un tel outil de manière aussi laconique.

Par ailleurs, le choix du module d'analyse va immédiatement engendrer la conception du deuxième module de cette fondation, le module de commercialisation des outils. Il résulte principalement de l'hétérogénéité des outils mentionnés précédemment. Et il est nécessaire d'avoir un moyen permettant de différencier, d'une manière aussi objective et scientifique que possible, les outils afin d'en sélectionner uniquement les plus pertinents. Surtout, c'est également à travers le moteur de jeu que nous pouvons clairement illustrer ce phénomène. Toutefois, le choix d'un ou de plusieurs moteurs de jeu dans une analyse semble être un processus particulièrement complexe. Une simple consultation de la liste des moteurs sur le site *IndieDB* indique pas moins de 574 moteurs de jeu répertoriés en 2023 (IndieDB 2023). Or, notre objectif principal est de pouvoir choisir un moteur potentiellement représentatif de l'évolution de l'industrie vidéoludique ainsi que des différentes générations de consoles. Et, puisque l'objectif principal d'un moteur de jeu est de concevoir des jeux, nous estimons qu'il est essentiel de lister les multiples jeux vidéo commercialisés conçus sur le moteur en question.

En utilisant notre critère au sein de cette même liste, nous avons la possibilité d'établir une comparaison entre deux moteurs : *Blender Game Engine* et *Unreal Engine 4*. Puisque notre problématique est centrée sur les jeux des différentes consoles *PlayStation*, nous pouvons vérifier l'existence de jeux publiés sur au moins une des cinq consoles de salon *PlayStation*. Et nous constatons également, grâce au site *Blender*, que « le *Blender Game Engine* a été supprimé » (Blender 2019 TDR[186]) du logiciel *Blender* depuis la mise à jour de la version 2.8. Or, nous ne trouvons aucune mention d'un quelconque jeu utilisant le *Blender Game Engine* sur l'une des consoles *PlayStation*. En revanche, on peut

185. « we should probably reserve the term "game engine" for software that is extensible and can be used as the foundation for many different games without major modification » (Gregory 2019, 12).

186. « the Blender Game Engine was removed » (Blender 2019).

rapidement constater l'utilisation du moteur *Unreal Engine 4* pour le développement et la publication d'un grand nombre de jeux, et nous pouvons au moins citer *Fortnite* (Epic Games 2017, PC).

C'est pourquoi nous définissons de manière limpide le deuxième critère de sélection comme étant l'usage du moteur dans au moins un projet commercial d'une machine étudiée dans notre recherche, que nous pourrions citer de manière indubitable. Grâce à ce critère, nous pouvons renforcer notre passerelle technologique infrangible puisque notre étude reste en lien avec l'industrie, sauf lorsque nous souhaitons évoquer sciemment des outils non utilisés par l'industrie. Toutefois, un regard néophyte sur ce sujet pourrait pourtant considérer la question de la publication de projets commerciaux utilisant un moteur comme étant des vétilles dans le contexte d'une étude scientifique. Un tel argument pourrait même encourager une approche non discriminante entre les multiples outils informatiques, les considérant *in fine* équivalents, en usant d'une hypothétique objectivité. Cependant, cela nous semblerait extrêmement peu rigoureux de procéder ainsi. Car l'objectif premier de notre étude est de pouvoir développer une approche scientifique pouvant attester de la représentativité d'un outil informatique dans l'industrie. Or, un moteur de jeu ne pourrait être représentatif uniquement dans le cas où il a déjà été utilisé dans le cadre du développement d'un jeu vidéo commercial.

Enfin, nous pouvons désormais élaborer le troisième module, que nous nommons le module de rationalisation. L'objectif de celui-ci est de pouvoir distinguer la manière de réaliser des analyses à la fois dans une étude synchronique, c'est-à-dire portant sur une seule génération de consoles, et diachronique, ou étudiant plusieurs générations, comme c'est le cas pour notre recherche. Sur ce plan, nous estimons qu'il existe deux possibilités :

- ► Une rationalisation abstraite, portée sur la théorie des outils de développement, préférable pour une recherche diachronique ;
- ► Une rationalisation concrète, analysant des scripts précis de programmation, adaptée à une recherche synchronique.

Dans notre étude, une rationalisation abstraite est plus que nécessaire, elle nous semble même obligatoire au regard de la période analysée, portant sur cinq générations différentes, en plus d'une légère analyse de la quatrième génération en guise de préambule. Il nous semble donc difficile de pouvoir traiter de manière

adéquate des aspects technologiques très spécifiques sur une échelle temporelle aussi longue, sans risquer de prolonger excessivement l'étude.

Finalement, nous avons constaté que ces critères permettent d'assurer d'une cohésion et solidité de notre passerelle technologique infrangible. Néanmoins, une interrogation majeure persiste, notamment au regard de l'aspect temporel de cette étude : de quelle manière est-il possible de traiter des outils anciens, en particulier pour une étude diachronique ? Cet aspect est précisément traité par le deuxième fondement qui fait l'objet de la prochaine sous-section.

2-1-B. OBSOLESCENCE INÉLUCTABLE

Après l'étape de validation décisive sur le choix des outils de développement à étudier, nous pouvons désormais présenter le deuxième fondement de la passerelle technologique infrangible, que nous nommons obsolescence inéluctable. Celui-ci est spécifiquement élaboré pour répondre aux multiples enjeux d'une étude diachronique. En effet, réaliser une étude sur des évolutions technologiques de la création de jeux vidéo implique *de facto* des outils potentiellement inaccessibles et technologiquement obsolètes. Certains outils informatiques disparaissent inéluctablement, laissant parfois des traces imperceptibles plutôt qu'indélébiles, comme nous l'observerons à travers plusieurs exemples. Nonobstant cet aspect presque inhérent à l'informatique, il est nécessaire d'apporter des solutions rigoureuses pour notre nouvelle approche. Celles-ci répondent pleinement aux spécificités de notre problématique, ce qui signifie qu'elles doivent être élaborées *ex nihilo* pour être cohérentes et pertinentes. C'est pourquoi nous avons établi trois préceptes concernant notre obsolescence inéluctable :

- Précepte d'inaccessibilité des outils ancestraux ;
- Précepte d'analyses technico-discursives pour le corpus développemental indirect ;
- Précepte de comparaison anachronique.

Tout d'abord, le premier précepte d'inaccessibilité des données ancestrales est principalement lié aux recherches entre la cinquième et la septième génération de consoles. Dans ces anciennes générations, la plupart des outils mentionnés sont aujourd'hui indisponibles et *in fine* inanalysables. Cet état de fait implique inexorablement de formuler des suppositions sur des outils anciens ou ancestraux, en se basant sur des éléments discursifs plutôt que sur des analyses purement techniques. Pour démontrer cette difficulté, nous allons illustrer ce phénomène via plusieurs cas, en débutant par celui des anciennes versions du moteur *Unreal Engine*. Il s'agit plus précisément de la version que nous évoquerons lors du quatrième chapitre, *Unreal Engine 3* et son édition publique UDK, aussi nommée *Unreal Development Kit*. Ceux-ci sont incompatibles avec une analyse directe puisque ces moteurs ne sont plus distribués officiellement par *Epic Games.* Le site officiel du moteur indique, en

effet, que l'entreprise « n'offre plus de support pour UDK, et recommande de commencer de nouveaux projets (...) avec UE4 » (Epic Games s. d.-a TDR)[187]. Naturellement, on pourrait rétorquer que le moteur UDK pourrait être téléchargeable sur certains sites d'archivage. Néanmoins, une telle action nous semble être en dehors du cadre éthique que nous souhaitons imposer à notre approche, et nous préférons utiliser exclusivement des ressources officielles. Ainsi, nous nous référons à la possibilité de télécharger un outil seulement s'il est disponible chez l'éditeur ou le développeur, afin de s'assurer non seulement qu'il s'agit d'une version adéquate, mais également légale d'un outil.

Nonobstant cette indisponibilité, on pourrait également rétorquer que de multiples outils peuvent, dans certains cas, être de nouveau disponibles à travers des sites officiels, par exemple via le dépôt d'une archive officielle. Et, à ce titre, on pourrait évoquer avec Claytone Crooks II l'exemple du logiciel de modélisation 3D *Gmax*, qui est un « logiciel de modélisation 3D et d'animation (...) développé par Discreet (...) [et qui] se compare avantageusement à 3ds max aussi bien en termes de fonctionnalités qu'au niveau du design » (Crooks II 2003, 2 TDR)[188]. Plus précisément, ce logiciel était souvent décrit comme étant, pour reprendre la formulation de Maarten Goldstein, « une version gratuite de 3D Studio Max (...) pour les joueurs/ les développeurs de *mod* » (Goldstein 2001 TDR) [189]. Officiellement disponible en 2019 sur le site de *TurboSquid*, ce même logiciel fut retiré par la suite, où la page indique dorénavant que *Gmax* « n'est plus un produit pris en charge par le site de *TurboSquid* ni par *Autodesk parce qu'ils* l'ont abandonné » (TurboSquid s. d. TDR)[190]. Ce fait sur *Gmax* démontre bien qu'une archive, même officielle, ne pourrait nullement être une solution pérenne pour l'utilisation d'un logiciel. Il est donc possible de constater la disparition d'un outil, même après avoir été rendu disponible via une archive officielle.

Alors, il serait possible d'évoquer un autre cas de figure, celui de la mise à disposition du code source d'un logiciel ou d'un moteur de jeu en *Open Source*.

187. « no longer support UDK, and recommend beginning new projects (...) using UE4 » (Epic Games s. d.-a).
188. « 3D-modeling and animation package (...) developed by Discreet (...) [and which] compares favorably to 3ds max in both functionality and design » (Crooks II 2003, 2).
189. « a free version of 3D Studio Max (...) for game players/ mod developers » (Goldstein 2001).
190. « is no longer a supported product through the TurboSquid website nor through Autodesk as they have discontinued it » (TurboSquid s. d.).

Nous pouvons définir le code source comme étant, d'après Justice, « le texte d'un programme tel qu'initialement écrit par les développeurs » (Justice 2021, 138 TDR)[191]. Et on pourrait définir, grâce à Marc Gregoire, que le « sens général du terme open source est que le code source est disponible et consultable par n'importe qui » (Gregoire 2021, 155 TDR)[192]. C'est pourquoi le terme *Open Source* s'oppose à *Closed Source*, un terme signifiant logiquement le contraire, c'est-à-dire un logiciel dont le code source est indisponible. Néanmoins, cette définition de la licence *Open Source* est très générique, alors qu'il existe des nuances importantes pour cette appellation, pouvant parfois générer des confusions. Nous n'entrerons point dans les aspects juridiques ou techniques de cette définition, ni dans les multiples variations, mais on citera simplement la variante nommée *Source Available*. Il s'agit d'une appellation plus restrictive, comme rappelé par Richard Fontana, puisqu'elle est « conçue (...) pour limiter ou empêcher (...) d'offrir des services concurrents basés sur les mêmes projets » (Fontana 2022, ch. 20.6 TDR)[193], autrement dit sur le même code.

Pour illustrer ces points, *Godot* est un logiciel *Open Source* puisqu'il est décrit comme étant un « moteur de jeu 2D et 3D communautaire, libre et open source » (Godot community 2024, Index TDR)[194]. En revanche, le moteur *Unreal Engine 4* est *Source Available*, puisqu'il est possible de « l'étendre, le modifier, en faire un *fork*, ou l'intégrer avec d'autres logiciels ou bibliothèques (...) [alors que l'on] ne peut pas combiner le code d'Unreal Engine avec du code couvert par un accord de licence de type "Copyleft" » (Epic Games s. d.-a, TDR)[195].

C'est ce fait qui permet d'introduire notre deuxième précepte pour répondre à cette problématique majeure, celui d'une analyse technico-discursive, où l'objectif serait plutôt d'analyser les technologies d'une manières discursive et technique lorsqu'elle n'est plus disponible. Sur cet aspect, nous pourrions

191. « the text of a program as originally written by developers » (Justice 2021, 138).
192. « general meaning of open-source is that the source code is available for anyone to look at » (Gregoire 2021, 155).
193. « designed (...) to limit or prevent (...) from offering competing services based on the same projects » (Fontana 2022, ch. 20.6).
194. « free and open source community-driven 2D and 3D game engine » (Godot community 2024, Index).
195. « extend it, modify it, fork it, or integrate it with other software or libraries (...) [while one] can't combine the Unreal Engine code with code covered by a "Copyleft" license agreement » (Epic Games s. d.-a).

justement citer l'entreprise *The Game Creators* qui a décidé, pour son moteur, de « rendre *DarkBasic Professional Open Source* et le mettre à disposition de la communauté pour qu'elle puisse le faire évoluer et le contrôler » (The Game Creators s. d. TDR)[196]. Or, nous pourrions soulever un écueil important sur l'utilisation d'un outil ancestral *Open Source* : l'analyse d'un outil qui, potentiellement, ne refléterait aucunement l'état actuel de la technologie de l'industrie vidéoludique. Pourtant, notre objectif est bien d'ancrer notre recherche, autant que possible, dans les préoccupations actuelles de l'industrie, où nous pouvons évoquer systématiquement les outils contemporains. Une analyse technique d'anciens outils semblerait donc contradictoire avec notre passerelle technologique infrangible, qui requiert justement une analyse technico-discursive pour des outils anciens, plutôt qu'une analyse purement technologique.

De surcroît, *The Game Creators* évoque l'idée de moderniser, ou de faire évoluer, le code source, ce qui pourrait s'apparenter à un procédé informatique nommé archéologie logicielle, qu'il serait intéressant d'évoquer succinctement. Celui-ci est surtout axé sur la production de mises à jour d'anciennes technologies. Et on pourrait avant tout préciser, grâce aux travaux d'Andy Hunt et de Dave Thomas, que le « code devient du code obsolète presque dès qu'il est écrit » (Hunt et Thomas 2002, 22 TDR)[197], signifiant *de facto* que l'obsolescence d'un code informatique est quasiment instantanée. Néanmoins, si nous souhaitons mieux circonscrire la définition de l'archéologie logicielle, nous pouvons indiquer, grâce à Pérez-Castillo, García-Rodríguez de Guzmán et Piattini, que « le processus d'archéologie logicielle consiste à analyser différents artefacts logiciels au moyen de techniques et d'outils de rétro-ingénierie » (Pérez-Castillo, García-Rodríguez de Guzmán et Piattini 2011, 87 TDR)[198]. De surcroît, l'objectif final d'un tel procédé est, comme indiqué par Hunt et Thomas, de pouvoir « comprendre comment et pourquoi ils ont écrit le code d'une certaine

196. « open source DarkBasic Professional and give it to the community to evolve and control » (The Game Creators s. d.).
197. « code becomes legacy code just about as soon as it's written » (Hunt et Thomas 2002, 22).
198. « the software archeology process consists of analyzing different software artifacts by means of reverse engineering techniques and tools » (Pérez-Castillo, García-Rodríguez de Guzmán et Piattini 2011, 87).

manière » (Hunt et Thomas 2002, 24 TDR)[199]. Cela signifie en substance que l'archéologie logicielle répond surtout à des besoins de modernisation de logiciels anciens. Or, l'objectif de notre recherche n'est nullement d'étudier la modernisation d'un logiciel ou d'un outil ancien, mais d'analyser exclusivement l'évolution d'outils de l'industrie vidéoludique à l'aune d'outils actuels. C'est pourquoi nous pouvons affirmer à la fois que l'étude d'outils ancestraux *Open Source* ainsi que l'archéologie logicielle ne répondent aucunement aux critères énoncés jusqu'à présent.

Au-delà de ce procédé de modernisation du code, nous pouvons évoquer un autre type de logiciel de création existant, que nous nommons les outils anachroniques de conception. Ces outils permettent de développer des jeux vidéo sur d'anciennes consoles, et sont généralement conçus bien après la fin de production desdites consoles, et destinés à des non-professionnels. Nous pouvons notamment citer le cas de *8bitworkshop*, un EDI (ou IDE en anglais) qui permet de développer des jeux sur d'anciennes consoles comme la *NES*, autrement dit de pouvoir, d'après Hugg « écrire du code en C ou en assembleur pour du matériel basé sur le 6502 ou le Z80 » (Hugg s.d. TDR)[200]. Définissons, au passage et grâce à Liang *et al.*, un EDI comme étant un « logiciel qui fournit un environnement de développement intégré (EDI) pour développer rapidement des programmes en C++ » (Liang *et al.* 2014, 39 TDR)[201], même s'il peut être utilisé pour d'autres langages de programmation que le *C++*. On élude néanmoins dans notre étude les éditeurs de texte et la différence avec des EDI.

Quoi qu'il en soit, si ces outils anachroniques de conception peuvent avoir des applications variées dans certaines études diachroniques particulières, ils se révèlent totalement inadaptés pour notre approche du DLT. En effet, ils compromettent la passerelle technologique infrangible puisqu'ils sont conçus a posteriori, et n'intègrent que de manière artificielle les multiples limitations inhérentes des consoles et outils originels. En d'autres termes, les

199. « understanding how and why they wrote the code the way they did » (Hunt et Thomas 2002, 24).
200. « write code for 6502 or Z80-based hardware in C, assembler » (Hugg s. d.).
201. « software that provides an integrated development environment (IDE) for rapidly developing C++ programs » (Liang *et al.* 2014, 39).

environnements simulant des outils de développement pour d'anciennes consoles sont incompatibles avec notre approche.

C'est pourquoi nous formulons clairement un dernier précepte impliquant une comparaison anachronique. Celui-ci a pour objectif de consolider la passerelle technologique infrangible grâce à une réalisation systématique de comparatifs entre les outils de développement des générations précédentes et les outils actuels. Par exemple, une analyse dans la cinquième génération de consoles pourra d'abord porter sur des informations du premier moteur *Unreal Engine*. Mais elle pourra également réaliser un comparatif avec le moteur *Unreal Engine 4*. L'intérêt est de pouvoir à la fois réaliser une analyse diachronique, de surcroît comparative, mais également de continuer à ancrer notre recherche dans les technologies les plus récentes possibles.

En clair, nous considérons qu'une étude diachronique dans le développement logiciel permet d'appréhender l'évolution des outils. Cependant, il n'est nullement possible d'utiliser ces mêmes outils, et l'histoire réelle devient inaccessible. Il ne reste donc qu'une seule possibilité : s'arc-bouter sur les éléments discursifs d'anciens développeurs, ce qui est justement traité par notre prochaine sous-section.

2-1-C. VÉRIDICTION TECHNICO-DISCURSIVE

Au regard du précepte des analyses technico-discursives du deuxième fondement, nous constatons que les aspects discursifs sur les outils informatiques sont essentiels dans une analyse diachronique portant sur la création de jeux vidéo. C'est pour cette raison que nous développons le troisième et dernier fondement de cette passerelle technologique infrangible, que nous nommons véridiction technico-discursive. D'emblée, précisons que l'objectif de ce fondement n'est nullement de réaliser des analyses sur le discours des développeurs, mais uniquement de vérifier la véridiction des arguments technologiques d'un discours. Nous souhaitons surtout faire usage de multiples théories sémiotiques sur les outils de développement eux-mêmes, et non pas sur l'analyse du discours. En d'autres termes, nous estimons que l'analyse discursive doit représenter une partie infime, voire infinitésimale, et exclusivement servir des analyses techniques ou technologiques. Car celle-ci n'est qu'une nécessité, une contingence pour notre étude diachronique, imposée par les problèmes soulevés à travers l'obsolescence inéluctable. De surcroît, notre problématique porte principalement sur les limitations dans la *création* des mécanismes interactifs, et non pas sur les *discours* sur la conception du *gameplay*.

Remarquons également que, si nous avons utilisé le terme de *véridiction* plutôt que celui de *véracité* ou *vérité*, c'est simplement pour souligner l'impossibilité de certifier ou valider la véracité des éléments discursifs. Nous ne pouvons pas rendre compte d'une quelconque *vérité*, mais seulement confirmer la plausibilité hypothétique des propos sur le plan technique. Ce même concept sémiotique de véridiction a été développé par Greimas, grâce à « la théorie saussurienne [qui] a contraint la sémiotique à inscrire (...) ses préoccupations (...) sur celui du dire-vrai, de la véridiction » (Greimas et Courtés 1993, 417). C'est pourquoi nous pouvons porter notre attention exclusivement sur un « discours-énoncé [qui] s'affiche comme vrai ou faux, mensonger ou secret » (*ibid.*) plutôt que sur la détermination de sa véracité. On inscrira donc, grâce à Eric Landowski, les propos sur des outils de développement dans « une problématique du faire croire vrai » (Landowski 2022, 256), même si nous tenons à modifier, quelque peu, la nature intrinsèque de ce concept de véridiction à travers ce troisième fondement.

Par ailleurs, cette perspective de véridiction provient d'un aspect que nous avons déjà mis en relief en introduction, mais de manière relativement succincte : un risque important de s'arc-bouter sur des discours fallacieux et pseudo technologiques, qui ne recouvrent aucune réalité technique dans l'informatique. Et, en plus des exemples déjà mentionnés en introduction, nous pouvons illustrer à nouveau ce phénomène via un argumentaire pour un jeu vidéo qui s'appuie, selon Therrien, sur une technologie « ''omni-dimension 4D'' (...) très suspecte » (Therrien 2012, 13 TDR)[202] et qui, à notre connaissance, n'existe pas. Cet état de fait semble inhérent au développement de jeux vidéo, potentiellement induit par une inaccessibilité de certaines technologies pour le grand public.

Dans tous les cas, ce troisième fondement doit être articulé autour de plusieurs axes. C'est pourquoi nous allons développer trois critères inédits pour notre véridiction technico-discursive, pouvant *in fine* valider la plausibilité d'un discours technique. Précisons que nous ne hiérarchisons aucunement ces différents critères, surtout élaborés pour fonctionner en synergie lors d'une analyse de la véridiction technico-discursive d'un énoncé donné. Nous pouvons désormais énumérer ces trois critères :

- ► Un énoncé technologique avéré et vérifiable ;
- ► Une expertise technique de l'énonciateur ;
- ► Une authenticité du support de l'énoncé.

Nous allons débuter par présenter le concept d'un énoncé technologique avéré et vérifiable, qui possède deux sous-objectifs majeurs : vérifier l'existence des termes techniques d'une part, et confirmer la disponibilité publique d'un outil capable de reproduire, *mutatis mutandis*, les concepts techniques d'un énoncé d'autre part. Nous ne vérifierons aucunement le référent, c'est-à-dire les outils originaux utilisés, mais seulement la logique technique du discours ainsi que sa potentielle réalisation. Sur ce point, Landowski évoque bien l'idée d'une véridiction qui « s'attache à rendre compte de la (...) grammaire — actantielle, modale, narrative, énonciative — qui préside aux stratégies de persuasion » (Landowski 2022). On s'inspire donc de cette idée de *grammaire* pour jauger la

202. « ''omnidimension 4-D'' (...) highly suspicious » (Therrien 2012, 13).

cohérence, voire la plausibilité technologique d'un énoncé sur le plan lexical et informatique.

Par ailleurs, c'est ici que nous différencions notre propre acception de la véridiction de celle de Joseph Courtés ou Greimas, en formulant un axiome de véridiction technologique. Sur ce point, Courtés va définir la véridiction comme représentant « les marques qui, dans (...) [les] discours, produisent comme effet de sens une impression de vérité » (Courtés 1991, ch. 1.2). Or, le deuxième sous-objectif de ce premier critère va au-delà de l'acception primaire de véridiction, pour vérifier la reproductibilité d'un concept technique sur un outil informatique. On s'oppose donc avec l'idée que « seul le langage est pertinent [et] la **réalité** paraît d'un autre ordre » (*ibid.*), et c'est ce qui justifie la juxtaposition des termes *technico* et *discursive* de l'appellation de notre troisième fondement. Ainsi, nous établissons plutôt ce troisième fondement comme étant une hybridité entre la véridiction et la reproductibilité scientifique via des outils informatiques.

De surcroît, l'usage de la véridiction implique que toutes les théories que nous allons élaborer en analysant des énoncés seront *de facto* purement hypothétiques. Elles ne pourront aucunement être substituables à des données factuelles provenant de l'analyse d'outils informatiques. Sur ce point, nous pouvons quantifier la véridiction d'une hypothèse à travers une variable inédite que nous nommons indice de véridiction technique. Partant de zéro, cet indice va s'incrémenter si les termes techniques sont réels mais aussi si l'aspect technique est reproductible par un logiciel ou outil informatique contemporain. Il oscillera donc entre zéro et deux, indiquant un indice de véridiction faible ou fort. L'indice de véridiction technique va surtout conditionner le respect du contrat de véridiction, concept développé par Greimas et qui « vise à établir une convention fiduciaire entre l'énonciateur* et l'énonciataire portant sur le statut véridictoire » (Greimas et Courtés 1993, 71). Cet indice peut donc être considéré comme étant la pierre angulaire du fondement de la véridiction technico-discursive, car il permet de jauger la crédibilité d'un énoncé.

Néanmoins, il est nécessaire d'analyser le deuxième critère pour renforcer la véridiction, que nous nommons expertise technique de l'énonciateur. Pour définir l'énonciateur, on peut s'appuyer sur Courtés qui indique la production d'un « "énoncé" (...) de la part de celui qui l'énonce (=l'"énonciateur") » (Courtés 2009, 19). L'objectif de celui-ci est de vérifier si cet énonciateur possède une expertise

technique dans le domaine de l'informatique et du développement de jeux vidéo 3D. Le fait de se référer ainsi à des développeurs, lorsque cela est possible, renforce implicitement le contrat de véridiction. Et c'est en particulier le cas pour des analyses très techniques qui exigent *de facto* des connaissances techniques relativement importantes. Par exemple, nous pouvons affirmer qu'un discours émanant d'un expert dans le développement de jeux vidéo 3D pourrait valider ce critère, alors qu'un discours journalistique ou d'un joueur ne permet aucunement de confirmer des hypothèses techniques. Naturellement, certaines sources ne permettent pas d'identifier clairement l'énonciateur. Dans un tel cas, on peut simplement le mettre en exergue en expliquant qu'il s'agit d'une expertise non identifiable, fragilisant *de facto* le contrat de véridiction.

Au-delà de cet aspect, il existe une particularité inhérente à l'étude scientifique portant sur les jeux vidéo, qui pourrait également rendre difficile l'analyse de l'énonciateur : l'éclectisme des sources. En effet, qu'il s'agisse d'ouvrages, d'articles, de manuels, de documentaires ou de publications en ligne, ceux-ci ne peuvent être entièrement constitués de productions purement scientifiques. Cette spécificité, comme nous le verrons tout au long de notre étude, provient non seulement d'une relation ambivalente des jeux vidéo avec le milieu universitaire, mais également de l'évolution très rapide des technologies. De plus, outre la complexité habituelle d'une recherche vidéoludique, notre étude implique l'intégration du processus et des données de développement de jeux. Ce qui, à notre connaissance, n'est que rarement abordé dans le milieu universitaire puisque, selon Martin, « bien que (…) certains auteurs abordent occasionnellement les industries du jeu, aucun (…) auteur n'a cela comme sujet principal de recherche » (Martin 2018 TDR)[203].

C'est justement ce qui justifie la raison d'être de notre troisième critère : l'authenticité du support de l'énoncé. Celui-ci implique de vérifier si le support de production de l'énoncé est authentique ou s'il présente des risques d'inauthenticité. Pour illustrer notre propos avec deux exemples opposés, nous pouvons d'abord citer le DVD *The Making of Metal Gear Solid 2: Sons of Liberty*

203. « while (…) authors may occasionally discuss the game industries, no (…) authors have this as their main research topic » (Martin 2018).

(Konami 2003, PS2), avec des énoncés produits par un programmeur devant une caméra. Le support est directement distribué par Konami, signifiant ainsi qu'il s'agit bien d'un document authentique. En revanche, nous avons également un commentaire sur internet, plus précisément sur le site de partage de liens *Hacker News*. Ce dernier concerne des aspects techniques d'une personne qui semble avoir, selon Malkia, « travaillé sur le portage de MGS sur PC vers la fin des années 1999-2000 » (Malkia 2015 TDR)[204], où il est question du jeu *Metal Gear Solid* (Konami 1998, PS1). Ce dernier peut être considéré comme étant impossible à authentifier, représenté par un pseudo anonyme, sans pour autant affirmer catégoriquement qu'il s'agit d'un commentaire authentique ou inauthentique. Et, par essence, la scientificité et *in fine* l'authenticité de ces matériaux discursifs est extrêmement variable, devenant dans certains cas difficilement conciliables avec une recherche universitaire qui se veut aussi rigoureuse. Or, si on se réfère au principe philologique de la sémiotique des textes d'après Rastier, « il faut établir le texte, le situer, par exemple en déterminant son authenticité » (Rastier 2010, 17).

Par ailleurs, nous pouvons mettre en surbrillance un nombre relativement varié de supports discursifs sur le développement de jeux vidéo : *Making-of*, *interviews*, documentaires, *vlog*, post*-mortem* ou bien d'autres encore. Dans les cas les plus rares, on peut également citer de véritables logiciels, comme l'illustre le « *Document of Metal Gear Solid 2*, un "*making-of*" interactif de MGS2 initialement publié en 2002 » (Wolf 2012a TDR)[205]. Précisons que celui-ci est différent du DVD *The Making of Metal Gear Solid 2: Sons of Liberty* (Konami 2003, PS2) cité plus haut, consultable comme un film. De surcroît, certaines informations pertinentes sont disponibles dans des forums ou des magazines parfois inaccessibles.

Il s'agit d'un phénomène généralisé puisque, selon Jennifer Whitson, « en raison du caractère confidentiel de l'industrie vidéoludique et (...) des secrets de fabrication, l'accès à ces sites est largement interdit » (Whitson 2018, 6 TDR)[206],

204. « worked on the port of MGS to PC back in 1999-2000 » (Malkia 2015).
205. « *Document of Metal Gear Solid 2*, an interactive "making of" of MGS2 originally released in 2002 » (Wolf 2012a).
206. « due to the secrecy of the game industry and (...) trade secrets, access to these sites is largely prohibited » (Whitson 2018, 6).

et les informations deviennent *in fine* également inaccessibles. Plus encore, cette difficulté est d'autant plus forte lors des débuts de la 3D, d'après Mark DeLoura, c'est-à-dire dans les années1990, période pendant laquelle « la programmation sur les consoles de jeu est (...) un art secret, les technologies ainsi que les APIs sont gardées secrètes via des accords de non-divulgation » (DeLoura 1999, 30 TDR)[207]. Une situation qui, comme nous le verrons par la suite, va évoluer, avec différents moteurs de jeu professionnels qui vont démocratiser l'accès au développement de jeux vidéo. Néanmoins, celle-ci implique parfois de pouvoir uniquement émettre des hypothèses, dont nous ne pourrons nullement vérifier la véracité.

Et, au-delà du secret pour les technologies générales de développement, on peut remarquer, grâce à Millington, que chaque équipe de développement « s'inspire de techniques (...), mais (...) les modifie au-delà de toute ressemblance, (...) rendant les algorithmes méconnaissables d'un studio à un autre » (Millington 2019, 3 TDR[208]). Cela a *in fine* une conséquence sur les théories puisque le développement de chaque jeu implique l'implémentation de fonctionnalités d'une façon presque unique. Par exemple, l'analyse d'un outil spécifique comme le moteur de jeu *Unreal Engine* ne pourrait aucunement permettre d'appréhender la manière dont un jeu a été réellement conçu. Concrètement, une analyse du discours des développeurs de *Tekken 8* (2024), qui est, d'après John Linneman, « le premier jeu de combat basé sur Unreal Engine 5 » (Linneman 2023 TDR)[209], permettra uniquement d'émettre une série d'hypothèses. Certes, celles-ci seraient plus conformes à la réalité du développement de ce jeu que s'il était question d'un moteur inaccessible. Néanmoins, nous restons dans des hypothèses puisqu'il existe une multitude d'implémentations possibles de chaque mécanisme interactif du jeu, comme nous le verrons par la suite.

L'utilisation d'outils identiques ne pourrait donc nous éclairer réellement sur l'ensemble des décisions, mais seulement permettre de former un spectre de

207. « game console programming is (...) a secret art, the technology and APIs are kept hidden by nondisclosure agreements » (DeLoura 1999, 30).
208. « draws techniques (...), but (...) modifies them beyond resemblance, (...) leaving algorithms unrecognizable from studio to studio » (Millington 2019, 3).
209. « the first fighting game based on Unreal Engine 5 » (Linneman 2023).

possibilités théorique, voire hypothétique. De surcroît, il n'existe qu'un seul support permettant de réellement comprendre le fonctionnement d'un logiciel : le code source. De plus, au-delà de l'indisponibilité de certains outils publiques, Brendan Sinclair souligne que « beaucoup de studios de jeux utilisent de toute façon des outils personnalisés en interne » (Sinclair 2014 TDR)[210], rendant plus difficile et opaque une analyse avec un moteur supposément identique.

En clair, nous avons conscience d'une objectivité scientifique relativement complexe à établir à travers la véridiction technico-discursive, mais nous estimons qu'il est nécessaire de se mesurer à cette difficulté pour pouvoir développer les solutions adéquates. Désormais, notre passerelle technologique infrangible semble suffisamment robuste pour pouvoir développer le générateur de théories vidéoludiques.

210. « lots of game studios use custom in-house tools anyway » (Sinclair 2014).

2-2. GÉNÉRATEUR DE THÉORIES VIDÉOLUDIQUES

La première section de ce chapitre fut consacrée à une présentation globale de notre approche sémio-informatique du DLT, avec les trois fondements de notre premier principe immuable, la passerelle technologique infrangible. À présent, il convient d'exposer de manière exhaustive le second principe immuable, que nous nommons le générateur de théories vidéoludiques. Comme esquissé au début de ce chapitre, il s'agit d'un outil fondamental pour notre thèse. Plus encore, il représente *in fine* l'objectif ultime de notre recherche puisqu'il permet de créer de nouvelles théories. Surtout, ces théories inédites créées *ex nihilo* (ou *from scratch* pour reprendre le jargon informatique) et *ex materia* (ou *out of the box*) seront inéluctablement ancrées dans l'industrie vidéoludique, notamment grâce au premier principe immuable. Naturellement, nous avons conscience, grâce à Landowski, que « l'architecture des modèles qu'on met au point dépend (...) des grilles de lecture auxquelles (...) on recourt au stade de l'observation intuitive initiale » (Landowski 2004, 57). C'est pourquoi nous souhaitons nous arc-bouter sur ce deuxième principe immuable pour diversifier les multiples grilles de lecture et étendre nos analyses.

Néanmoins, si notre DLT est une approche sémio-informatique, on peut mettre également en exergue trois sémiotiques appliquées qui peuvent se confondre avec notre approche, alors même qu'elles sont intrinsèquement différentes. C'est pourquoi nous allons décrire ces trois sous-domaines succinctement, à savoir la linguistique informatique, la sémiotique computationnelle ainsi que la sémiotique des jeux vidéo.

Dans le premier cas, la linguistique informatique est une étude de phénomènes langagiers à travers l'usage de multiples technologies. On peut la définir précisément comme étant, d'après François-Régis Chaumartin et Pirmin Lemberger, « l'ensemble des méthodes de traitement du langage qui permettent de "comprendre" automatiquement du texte pour en extraire du sens » (Chaumartin et Lemberger 2020, 64). Nous pouvons également perfectionner cette définition, grâce à Nicoletta Calzolari, en indiquant également qu'elle « inclut les disciplines traitant des modèles, méthodes, technologies, systèmes et applications concernant le traitement automatique d'une langue » (Calzolari

2009, 329 TDR)[211]. Or, notre problématique ne porte nullement sur le traitement automatique, ou informatisé, d'une langue naturelle humaine. Ainsi, il est possible d'affirmer que notre recherche ne présente aucun lien avec la linguistique informatique, sous quelque forme que ce soit.

Le deuxième cas est celui de la sémiotique computationnelle, que Jean-Guy Meunier va définir comme étant « l'application de théories sémiotiques aux systèmes informatiques et aux media digitaux interactifs » (Meunier 2022, 30 TDR) [212] . Néanmoins, il s'agit d'un sous-domaine qui semble être lié à l'informatique générale, et non pas au domaine très spécifique du développement de jeux vidéo en 3D temps réel. C'est notamment le cas de la plupart des théories citées et expliquées par Meunier (2022). Or, notre problématique n'est pas la sémiotisation générale de l'informatique en tant que domaine, mais l'utilisation de la sémiotique comme *outil* pour répondre à une problématique très précise, liée à un aspect particulier de l'industrie vidéoludique professionnelle. Nous partons surtout du point de vue que des théories ne seront jamais suffisamment rigoureuses si elles ne sont pas directement en relation avec le développement de jeux vidéo 3D, et si elles ne sont pas conçues spécifiquement dans ce cadre. Nous ne pouvons donc nullement prendre le risque d'une approximation induite par des théories trop générales, génériques ou trop en dehors de notre sujet.

De plus, la finalité semble être plus proche, pour reprendre notre distinction élaborée en introduction, de la sémiologie théorique plutôt que de la sémiotique appliquée, en dépit des apparences. En effet, Meunier indique que « le cadre conceptuel global de la sémiotique n'a pas vraiment intégré les théories informatiques classiques » (Meunier 2022, 31 TDR)[213], et cette segmentation implique hypothétiquement une absence de lien entre les industries de l'informatique et la sémiotique computationnelle. Or, nous avons bien mentionné en introduction notre objectif d'utiliser le cadre théorique de l'informatique, mais aussi de nous adresser à l'industrie vidéoludique. Par conséquent, nous

211 . « includes the disciplines dealing with models, methods, technologies, systems and applications concerning the automatic processing of a language » (Calzolari 2009, 329).

212. « the application of semiotic theories to computer systems and interactive digital media » (Meunier 2022, 30).

213 . « the overall conceptual framework of semiotics did not really integrate classical computational theories » (Meunier 2022, 31).

considérons que la sémiotique computationnelle est totalement incompatible avec notre approche sémio-informatique inédite du DLT, qui de plus est une sous-branche de l'informatique, et *in fine* une sémiotique appliquée, et non pas une sémiologie théorique.

Au regard du troisième cas, nous avons déjà longuement traité celui-ci en introduction, puisque la sémiotique du jeu vidéo peut généralement se confondre avec les *Game Studies*. Elle reflète une étude non-technique des jeux vidéo, autrement dit aux antipodes de notre recherche. Les types d'analyses, pour reprendre la nomenclature développée par Egenfeldt-Nielsen, Smith et Tosca (2015, 11) et présentée en introduction, seraient donc externes aux outils de création de jeux vidéo. Or, la nature intrinsèque de notre problématique est liée au développement de jeux vidéo, et au jeu vidéo comme étant un logiciel informatique, et non pas au jeu vidéo en tant que *media*.

Désormais, nous pouvons présenter trois éléments majeurs du générateur de théories vidéoludiques, que nous allons exposer dans les trois sous-sections correspondantes :

- ► Symbiose sémio-informatiques ;
- ► Interthéoricité logicielle ;
- ► Hyperstructure tridimensionnelle.

Le premier élément représente les multiples relations entre les deux disciplines sillonnant cette thèse, avec une mise en exergue de l'interdisciplinarité. Le deuxième est celui de l'interthéoricité, un outil majeur développé par Astrid Guillaume et permettant le « processus de création de théories qui réunit des éléments d'autres théories » (Guillaume 2014, 50). Enfin, le troisième est la constitution d'une hyperstructure 3D, l'objectif ultime du générateur de théories vidéoludiques. Nous pouvons à présent nous intéresser, plus encore, au premier élément du générateur.

2-2-A. SYMBIOSE SÉMIO-INFORMATIQUE

Les multiples interactions entre la sémiologie et l'informatique nous permettent de créer des théories riches et protéiformes, s'appuyant sur des concepts inédits et novateurs. Et, dans le cadre d'un alliage de ces deux domaines dans notre recherche, nous souhaitons développer des modèles permettant de mettre en exergue leurs caractéristiques intrinsèques et de créer *in fine* des théories bicéphales. C'est pourquoi nous estimons qu'il est nécessaire d'expliciter la relation entre la sémiotique et l'informatique à travers un modèle relationnel que nous utiliserons ici : l'interdisciplinarité.

Tout d'abord, cette question des liens entre différentes disciplines était assujettie à de nombreux débats au moins depuis, selon Patrick Charaudeau, « les années 70-80 (...) [où] il était alors délicat de parler d'interdisciplinarité, de pluridisciplinarité ou de transdisciplinarité, (...) [au] risque de se voir dénier le label de scientificité » (Charaudeau 2010, 2). Pourtant, et d'après Paul Orianne, l'objectif de faire converger plusieurs disciplines, ou de pouvoir au moins créer une communication entre elles, s'appuyait surtout sur « un même refus, fondé sur le soupçon qu'une discipline, à elle seule, (...) [ne pourrait] donner du réel qu'une image tronquée ou déformée » (Orianne 1983, 39). Une vision que nous partageons particulièrement dans le cadre d'une analyse des outils de création de l'interactivité dans les jeux vidéo. De plus, au regard de notre problématique, il nous semble difficile de réaliser une étude monodisciplinaire puisque, comme souligné par Charaudeau, « aucune discipline (...) ne peut rester enfermée dans son orthodoxie » (Charaudeau 2010, 19). De surcroît, on peut également constater, grâce à Guillaume, que « quand les théories sont monodisciplinaires, elles évoluent peu » (Guillaume 2014, 17). Or, pour innover et sortir des normes habituelles d'une discipline, nous estimons également qu'il est nécessaire d'intégrer dans une même étude d'autres disciplines.

Et c'est grâce à cet objectif que, selon Charaudeau, et toujours « dans les années 70-80, (...) Roland Barthes, Pierre Bourdieu, Michel Foucault, Gérard Genette, Edgar Morin, Paul Ricœur (...) ont ouvert (...) la voie à une multiplicité de rapports entre les disciplines » (Charaudeau 2010, 19). Nous pouvons donc inscrire notre recherche dans la continuité de ce même esprit, grâce à Guillaume, avec des « disciplines scientifiques [qui] (...) semblent se décloisonner de plus en

plus en ce début de XXI[e] siècle pour aller vers plus d'interdisciplinarité, de pluridisciplinarité et de transdisciplinarité » (Guillaume 2014, 48). Surtout, nous estimons également qu'un « décloisonnement des disciplines et la contrastivité s'imposent comme méthode pour comprendre la totalité d'un problème dans le temps et sur les temps » (*ibid.*). C'est donc par ce même phénomène qu'il nous est possible d'appréhender l'interactivité des univers virtuels et l'élaboration des mécanismes.

Toutefois, on pourrait interroger la manière dont nous devons décloisonner puis agencer ces deux disciplines. Pour cela, nous pourrions citer cinq modèles principaux pour lier les disciplines entre elles, en nous basant sur l'article de Charaudeau (2010) : interdisciplinarité, monodisciplinarité, multidisciplinarité, pluridisciplinarité ou encore transdisciplinarité. Sans définir précisément chaque modèle de décloisonnement des disciplines, puisque cela n'est pas l'objet de notre étude, nous estimons que nos deux disciplines doivent tout de même conserver une certaine délimitation. C'est pourquoi nous décidons de choisir, pour reprendre la définition de Catherine Carlo, une « voie moyenne (...) celle de l'interdisciplinarité, qui établit des connexions plus ou moins denses, au plan méthodologique ou conceptuel entre les disciplines » (Carlo 2015, 2). Ce même modèle est une méthodologie qui, comme rappelé par Charaudeau, « échange, coopère, partage en interrogeant et intègre de façon critique » (Charaudeau 2010, 19) les données des différentes disciplines. Néanmoins, on constate, grâce à Édouard Kleinpeter, que cette même interdisciplinarité « n'a pas de définition fixe dans la littérature » (Kleinpeter 2013, 126). C'est pourquoi notre objectif est simplement d'adapter l'interdisciplinarité à la nature intrinsèque de la problématique de notre recherche ainsi qu'aux deux domaines impliqués.

Par conséquent, cette interdisciplinarité implique de développer des liens multiples entre les aspects informatiques et sémiotiques de notre thèse, et de les réguler en fonction de concepts robustes. Cela permettra *in fine* de structurer la création de théories dans notre thèse. Nous choisissons ainsi d'élaborer de multiples types d'oppositions entre les concepts, en nous basant principalement sur Courtés (1991) et sur le carré sémiotique ou véridictoire de Greimas (Greimas et Courtés 1993). Et, pour rapprocher ces relations de notre problématique, nous pouvons souligner que cette dernière traite certes des limitations potentiellement induites par les genres vidéoludiques, mais analyse surtout les relations entre les

multiples mécanismes interactifs. Car le fait de confirmer ou d'infirmer l'existence des multiples catégories de *gameplay* dans les outils de création de jeux vidéo revient inéluctablement à analyser des relations entre les mécanismes interactifs. Nous pouvons ainsi lister les trois types de relations théoriques que nous allons définir :

- ► Oppositions catégorielles ;
- ► Oppositions graduelles ;
- ► Relations de complémentarité ou relations de combinaison.

Ainsi, nous pouvons désormais définir, grâce à Courtés, « les oppositions dites catégorielles (vrai/ faux ; légal/ illégal) qui n'admettent pas de moyen terme » (Courtés 1991, 2.1), pertinentes pour opposer les différents outils de création des mécanismes interactifs dans les moteurs de jeu ou langages de programmation. Surtout, l'intérêt principal des oppositions catégorielles est de pouvoir formuler de multiples catégories d'outils de création du *gameplay*. Cela permet de vérifier si nous retrouvons *in fine* une nomenclature qui peut s'apparenter aux genres vidéoludiques, mais en partant d'un prisme purement technologique. Néanmoins, identifier des oppositions claires et catégoriques pourrait s'avérer difficile, et risquerait de simplifier excessivement la nature intrinsèque des mécanismes interactifs, voire d'en occulter certains aspects.

Ainsi, c'est la raison pour laquelle nous ajoutons les oppositions graduelles, pouvant affiner certaines oppositions. Nous pouvons également définir, pour reprendre les propos de Courtés, ces mêmes « oppositions graduelles (...) sous forme scalaire, avec autant de positions intermédiaires possibles » (Courtés 1991, 2.1). Ces oppositions nuancées sont particulièrement intéressantes car elles entrent hypothétiquement en contradiction avec la nomenclature purement catégorielle des genres vidéoludiques. C'est pourquoi l'identification d'outils pour créer des mécanismes interactifs possédant des aspects graduels ou scalaires pourrait indéniablement répondre à notre problématique. Néanmoins, ces deux types d'oppositions ne permettent toujours pas de mettre en exergue la combinaison des mécanismes interactifs. Or, il s'agit bien d'une préoccupation centrale puisqu'elle est mentionnée dès la première page de notre introduction.

C'est pourquoi nous pouvons désormais définir les relations de complémentarité ou de combinaison, qui s'inspirent du carré sémiotique de Greimas. Mais, avant de circonscrire ces oppositions, nous devons définir le

carré sémiotique, qui est « la représentation* visuelle de l'articulation logique d'une catégorie* sémantique quelconque (...) entre au moins deux termes* » (Greimas et Courtés 1993, 29). Fontanille va relever les points centraux de cet outil, décrivant celui-ci comme étant « la réunion de deux types d'oppositions binaires dans un seul système » (Fontanille 2006, 29 TDR)[214]. Et, pour définir les relations de complémentarité, on s'inspire surtout de « l'opération d'implication établit entre les termes ($\overline{A}$ et non-A) et ($\overline{\text{non-A}}$ et A) [qui] est une relation de **complémentarité** » (Greimas et Courtés 1993, 31). Les relations de complémentarité ou de combinaison sont donc au cœur des jeux en monde ouvert, comme indiqué en introduction. C'est pourquoi cette opposition est fondamentale car elle relève d'une contradiction par rapport aux genres vidéoludiques.

Ainsi, nous estimons que ces trois types de relations sont suffisants pour répondre à la problématique de notre thèse, puisque nous pouvons mettre au jour les relations entre les différents mécanismes interactifs. Néanmoins, ces oppositions se bornent surtout à analyser la création des mécanismes interactifs. Or, nous souhaitons également utiliser le concept d'interthéoricité pour intégrer harmonieusement différentes théories externes dans notre thèse et les reformuler de manière pertinente, ce que nous décrivons dans la prochaine sous-partie.

214. « the uniting of two types of binary oppositions in a single system » (Fontanille 2006, 29).

2-2-B. INTERTHÉORICITÉ LOGICIELLE

En complément des phénomènes relationnels que nous avons précisés précédemment, nous souhaitons désormais présenter un outil facilitant la réadaptation d'une théorie dans un autre domaine. À cet égard, nous estimons qu'il est nécessaire de procéder à un réajustement de l'intégralité des théories utilisées pour qu'elles puissent répondre aux multiples aspects de notre problématique. C'est pourquoi notre approche du DLT s'inscrit dans la lignée scientifique d'un concept majeur développé par Guillaume (2014) et nommé interthéoricité. Nous pouvons d'emblée définir ce concept comme étant, d'après Guillaume, « le transfert de théories d'une discipline à l'autre avec ou non des modifications d'objectifs » (Guillaume 2020, 127). Celui-ci est d'une importance capitale pour notre recherche puisqu'il « ouvre (...) la voie à une nouvelle manière de comprendre les théories d'hier et de créer celles de demain » (Guillaume 2014, 17), indispensable pour notre acception de l'interdisciplinarité. Nous souhaitons particulièrement exploiter une certaine « élasticité des théories » (*ibid.*) pour mieux réadapter certaines théories pour le contexte du développement de jeux vidéo 3D. Surtout, Guillaume indique que « l'interthéoricité permet la création de nouvelles théories » (Guillaume 2020, 127), c'est pourquoi nous avons précédemment évoqué la création de théories *ex nihilo* ou *ex materia*.

De surcroît, ce processus permet d'imposer une rigueur intellectuelle et méthodologique substantielle, en partant du postulat que chaque théorie doit être potentiellement réajustée pour devenir pertinente dans notre cadre théorique. C'est également la raison principale expliquant pourquoi nous estimons, grâce à Renée Bouveresse, que « aucune théorie ne peut être considérée comme établie (...) elle n'est jamais qu'une hypothèse susceptible d'être remise en cause » (Bouveresse 1978, 11). C'est pourquoi nous sommes d'accord avec les propos de Guillaume, indiquant que « l'interthéoricité permet des innovations théoriques par hybridation ou recomposition novatrices, qui impliquent de repenser les définitions et les approches scientifiques » (Guillaume 2020, 128). Nous pouvons ainsi élaborer une légère adaptation sémantique du concept de l'interthéoricité au domaine sémio-informatique, afin de le rendre harmonieux avec notre approche du DLT. Ainsi, nous développons le concept

d'interthéoricité logicielle qui subdivise le processus d'interthéoricité en deux opérations distinctes :

- ► Une exportation originelle ;
- ► Une importation théorique.

D'emblée, nous précisons que ce jargon provient directement, pour ces deux étapes, d'opérations du même nom pour incorporer des modèles 3D dans un jeu vidéo. Ces concepts sont relativement simples, mais nous permettent de traiter rigoureusement du processus d'interthéoricité. Ainsi, l'objectif de la première opération, que nous nommons exportation originelle, consiste à présenter la théorie de départ, sous sa forme originale, et à expliciter les intérêts d'exploiter une interthéoricité logicielle sur celle-ci. Précisons que nous ne nous occuperons nullement du contexte de création ou d'interprétation de ces mêmes théories de départ, ni même des éventuelles controverses de celles-ci. Ces aspects ne nous intéressent aucunement, et l'objectif de cette opération est simplement d'intégrer dans notre approche du DLT des théories externes qui peuvent faire progresser notre recherche.

Ensuite, nous allons réaliser la deuxième opération que nous nommons importation théorique, impliquant de modifier légèrement l'appellation de la théorie de départ, puis d'adapter son sens. Ce qui est justement décrit par Guillaume (2020) par plasticité, élasticité ou encore hybridité. L'intérêt est d'avoir une théorie qui puisse correspondre à l'industrie vidéoludique ou aux outils de développement de jeux vidéo. Cela permet d'ajuster celle-ci à notre problématique et d'enrichir substantiellement notre propos à travers des perspectives et prismes différents.

Désormais, nous avons présenté de manière limpide l'interthéoricité logicielle, permettant de développer l'ensemble des théories pour notre thèse. Néanmoins, celles-ci seront disparates, et chacune ne pourrait répondre à la problématique que de manière partielle. C'est pourquoi nous utiliserons un outil final qui va unifier l'ensemble de nos théories, et que nous allons présenter dans la sous-section suivante : l'hyperstructure 3D.

2-2-C. HYPERSTRUCTURE TRIDIMENSIONNELLE

L'objectif ultime de notre générateur de théories vidéoludiques, au-delà d'une symbiose sémio-informatique ainsi que de la création de théories inédites via le processus de l'interthéoricité logicielle, est de pouvoir unifier toutes ces théories dans une structure sémio-informatique, que nous nommons hyperstructure 3D. Celle-ci sera érigée au regard de notre problématique exposée *supra*, et formera un ensemble *sui generis* de toutes les théories élaborées dans notre thèse. L'intérêt est d'aller au-delà des contingences de chaque théorie pour présenter un modèle englobant la totalité des résultats de nos recherches ; une structure capable de mettre au jour la clef de voûte d'un outil de création de jeux vidéo en 3D temps réel. L'objectif majeur de cette hyperstructure 3D est similaire, pour reprendre la formulation de Chandler, à celui de la « sémiotique [qui] a une fonction de synthèse importante, cherchant à étudier la création de sens et la représentation » (Chandler 2022, 291 TDR)[215].

Par ailleurs, ce nom que nous attribuons à ce nouveau modèle sémiotique est lié à une considération bicéphale, avec à la fois une structure sémiotique, mais également d'une structuration complète des caractéristiques des logiciels en 3D temps réel, d'où l'idée du préfixe *hyper-*. Et, grâce aux précisions technologiques importantes de cette structure, mais aussi la prise en compte complète des contraintes technologiques présentes ou passées, on peut l'adapter à l'ensemble des microcosmes virtuels.

Nous pouvons donc également développer trois caractéristiques immanentes inédites qui renforceront la robustesse de notre hyperstructure 3D :

- ► Le type d'hyperstructure (interprétative ou générative) ;
- ► L'interface intemporelle ;
- ► L'implémentation générationnelle.

Ainsi, nous pouvons définir deux types distincts de notre hyperstructure 3D, relevant d'une opposition catégorielle mise en exergue par Nicolas Couégnas (2022) dans le cadre de plusieurs analyses : interprétative et générative. En effet, On peut d'abord décrire une « sémantique textuelle (...) interprétative,

215. « semiotics [which] has an important synthesizing function, seeking to study meaning-making and representation » (Chandler 2022, 291).

rétrospective (...) [portant] sur des énoncés réalisés » (Couégnas 2022). Si nous transposons cela pour notre approche sémio-informatique, nous pourrions évoquer l'idée d'une nature interprétative de notre structure qui porterait sur des jeux vidéo déjà produits, dont il faudrait seulement interpréter la création.

Or, il oppose ce prisme à celui d'une « sémiotique discursive (...) générative, [qui] se place du côté de la production » (*ibid.*). Il s'agirait alors d'étudier la conception de jeux, et plus précisément dans notre cas, les outils de création de jeux. C'est pourquoi notre structure peut donc être qualifiée d'hyperstructure 3D interprétative, avec l'analyse de la création d'œuvres existantes, ou d'hyperstructure 3D générative, étudiant des outils de conception en eux-mêmes.

Les deux dernières caractéristiques immanentes de notre hyperstructure sont l'interface intemporelle et l'implémentation générationnelle. Il s'agit d'une opposition de complémentarité qui utilise deux concepts que nous empruntons à la programmation informatique : l'interface et l'implémentation. Pour d'abord définir ces deux termes, on peut s'appuyer sur l'explication technique de Gregoire où une « implémentation est le code (...) permettant d'accomplir la tâche (...) [alors que] l'interface est la manière dont d'autres personnes utilisent (...) [le] code » (Gregoire 2021, 189 TDR)[216]. L'objectif de cette distinction est de pouvoir élaborer des caractéristiques intemporelles de notre structure, c'est-à-dire exploitables pour n'importe quelle génération de consoles, ainsi que certaines caractéristiques générationnelles, entièrement liées à une génération de consoles précise.

Pour illustrer cela à travers un exemple, nous pouvons hypothétiquement affirmer qu'une animation d'un personnage virtuel est intemporelle, et nous sommes donc en présence de la description d'une interface intemporelle. Or, les spécifications techniques pour l'animation d'un personnage, c'est-à-dire les techniques utilisées dans le moteur de jeu pour animer le modèle 3D d'un personnage, sont liées à une, voire plusieurs, générations de consoles. Il s'agit, dans ce second cas, d'une implémentation générationnelle, pouvant être liée à

216. « implementation is the code (...) to accomplish the task (...) [while] the interface is the way that other people use (...) [the] code » (Gregoire 2021, 189).

des générations précises et avec des aspects techniques que nous pourrons clairement identifier.

Naturellement, une implémentation générationnelle peut être similaire, voire identique, pour plusieurs générations de consoles. Mais il est toujours possible d'apporter des précisions ou des nuances sur les aspects techniques, alors que l'interface intemporelle n'a aucunement besoin de précisions techniques.

Désormais, nous avons présenté de manière exhaustive les multiples aspects de notre approche sémio-informatique du DLT, avec ses différents concepts transversaux à notre recherche. Nous pouvons à présent développer le troisième chapitre de notre thèse, qui va esquisser les prémices de la conception des moteurs de jeu en 3D temps réel.

2-3. SYNTHÈSE DU DLT

Ce deuxième chapitre a permis de présenter les aspects fondamentaux de notre approche sémio-informatique inédite, nommée Développement Logiciel Tridimensionnel (DLT) et élaborée spécifiquement pour les besoins de notre problématique. Cette innovation répond surtout aux limitations induites par l'approche générique du champ des *Game Studies* présenté en introduction ; un même champ qui occulte souvent la dimension technique dans les analyses d'œuvres vidéoludiques. C'est pourquoi les *Game Studies* semblent *de facto* incompatibles avec notre recherche portant sur les outils de développement de jeux, ne pouvant réellement répondre à notre problématique qui s'appuie sur des aspects exclusivement techniques. Ainsi, nous avons intégré à notre approche nouvelle du DLT une méthodologie de recherche indubitablement ancrée dans la technologie, et répondant *in fine* aux enjeux scientifiques et techniques de notre thèse.

Naturellement, si nous avons décidé de présenter celle-ci dès le deuxième chapitre plutôt qu'à la fin de notre thèse, c'était surtout dans l'optique de rendre intelligible notre propos par la suite, puisque nous allons nous référer aux multiples fondements du DLT dans la totalité des chapitres ultérieurs. Néanmoins, nous avons constitué ses différentes caractéristiques tout au long de la recherche et affiné cette méthode pour en assurer la scientificité, raison pour laquelle celle-ci ne pourrait nullement être placée en introduction. En substance, le DLT fait partie intégrante de la recherche. Par ailleurs, notre approche du DLT est constituée de deux principes immuables : la passerelle technologique infrangible et le générateur de théories vidéoludiques. Il s'agit de la raison pour laquelle nous avons fragmenté ce chapitre en deux sections distinctes décrites ci-dessous.

2-1. PASSERELLE TECHNOLOGIQUE INFRANGIBLE

Cette section majeure représente le premier principe immuable au nom éponyme, dont l'objectif est de créer une *passerelle* entre des analyses scientifiques portant sur la création de l'interactivité et des considérations technologiques de l'industrie vidéoludique. Ce principe est constitué de trois fondements distincts, au cœur de chaque sous-section : la fortification

technologique, l'obsolescence inéluctable et la véridiction technico-discursive. Un tel procédé est effectué par l'intermédiaire d'outils professionnels de développement de jeux vidéo.

Dès **l'introduction**, nous avons présenté la définition des jeux vidéo, que nous considérons comme étant exclusivement des logiciels informatiques interactifs (Savchenko 2000 ; Rouse III 2005). Néanmoins, nous avons estimé que les autres définitions (Galanina, Akchelov et Sakharova 2020) sont indubitablement pertinentes, mais pour d'autres études scientifiques, voire d'autres approches. Puis nous avons exposé la méthodologie scientifique que nous utiliserons dans notre approche, débutant par la problématisation, l'hypothèse hypothético-déductive puis la sélection du corpus, opérations réalisées dans notre chapitre introductif. Et, dans chaque première section des chapitres ultérieurs, on réalisera une revue de la littérature historique de l'industrie vidéoludique, puis scientifique et informatique. Ensuite, nous analyserons dans chaque deuxième section toutes les données historico-scientifiques, en mettant en exergue quelques exemples de nos multiples corpus, pour créer des théories *ex nihilo* ou *ex materia*.

Ainsi, le premier fondement dans la sous-section nommée **Fortification technologique** a permis de créer un lien entre une analyse théorique portant sur les technologies et des considérations professionnelles de l'industrie du développement de jeux vidéo 3D. Ce fondement est composé de trois modules distincts : module de précision technologique, module de commercialisation et module de rationalisation. Le premier implique de choisir un niveau précis d'analyse technologique comme le matériel, le moteur de jeu, le langage de programmation ou encore l'API 3D. Le deuxième module vise à déterminer si les outils de développement analysés dans notre recherche ont déjà été employés dans un contexte professionnel et commercial. Enfin, le troisième module a pour objectif de différencier, puis déterminer la pertinence, entre des analyses abstraites sur des technologies, et plutôt haut niveau pour reprendre des termes informatiques, ou concrètes et très précises, autrement dit bas niveau, portant essentiellement sur du code informatique.

Puis le deuxième fondement de la section, **Obsolescence inéluctable**, a contribué à résoudre des problématiques engendrées par une étude diachronique, portant dans notre cas sur d'anciens outils informatiques et

d'anciennes générations de consoles. Celui-ci est composé de trois préceptes : l'inaccessibilité des outils ancestraux, l'élaboration d'analyses technico-discursives et enfin la comparaison anachronique. Le premier va mettre en exergue l'inaccessibilité des outils de développement des anciennes générations, particulièrement de la cinquième à la septième génération. De ce premier précepte découle le deuxième, qui implique de réaliser exclusivement des analyses technico-discursives sur les anciens outils, c'est-à-dire de s'appuyer sur des discours techniques. Quant au troisième précepte, celui-ci permet de consolider le lien avec l'industrie vidéoludique, en réalisant systématiquement des comparatifs entre les outils de développement des générations précédentes et les outils actuels.

Enfin, le troisième fondement exposé dans la section **Véridiction technico-discursive** permet de circonscrire les analyses technico-discursives sur le corpus développemental et le corpus vidéoludique. Le concept de véridiction, que nous empruntons à Greimas (Greimas et Courtés 1993 ; Landowski 2022), est utilisé pour vérifier non pas la véracité d'un énoncé, mais seulement la plausibilité, ici sur le plan technologique. Précisons que notre acception de la véridiction est altérée par notre concept de l'axiome de véridiction technologique, où nous analysons également les aspects techniques d'un énoncé. Nous avons ainsi développé trois critères principaux pour notre véridiction : un énoncé technologique avéré et vérifiable, une expertise technique de l'énonciateur et enfin une authenticité du support de l'énoncé.

2-2. GÉNÉRATEUR DE THÉORIES VIDÉOLUDIQUES

La deuxième section majeure est consacrée au second principe immuable : le générateur de théories vidéoludiques, ayant pour objectif de permettre la création de nouvelles théories *ex nihilo* (ou *from scratch*) ou *ex materia* (ou *out of the box*). C'est ce second principe qui va permettre de concevoir la structure sémio-informatique ultime de notre approche du DLT : l'hyperstructure 3D.

L'introduction nous a permis de distinguer trois sous-domaines différents qu'on estime être totalement éloignés de notre approche : la linguistique informatique, la sémiotique computationnelle, et enfin la sémiotique des jeux vidéo. La première est liée à des préoccupations linguistiques (Chaumartin et Lemberger 2020 ; Calzolari 2009) que nous estimons sans rapport avec notre

problématique ; la deuxième, traitant de l'informatique dans sa globalité (Meunier 2022), nous paraît trop générale et générique, voire éloignée des spécificités de l'industrie vidéoludique, pourtant au cœur de notre approche ; la troisième se distingue de notre approche puisqu'elle analyse les jeux de manière non technique, et serait assimilable au champ scientifique des *Game Studies* (Egenfeldt-Nielsen, Smith et Tosca 2015). Or, nous ancrons plutôt notre étude dans le cadre théorique de l'informatique, et non pas dans celui des *Game Studies* ou du *Game Design*.

Puis, dans la première sous-section, **Symbiose sémio-informatique**, nous avons présenté le modèle principal de décloisonnement des disciplines utilisé pour lier la sémiotique et l'informatique : l'interdisciplinarité. Celui-ci se distingue des autres modèles, notamment la monodisciplinarité, la multidisciplinarité, la pluridisciplinarité ou encore la transdisciplinarité (Charaudeau 2010 ; Orianne 1983 ; Guillaume 2014 ; Carlo 2015 ; Kleinpeter 2013). Ensuite, nous avons formulé trois types de relations que nous utiliserons dans nos théories ultérieures : les oppositions catégorielles, les oppositions graduelles et enfin les relations de complémentarité ou de combinaison (Courtés 1991 ; Greimas et Courtés 1993 ; Fontanille 2006).

Puis la deuxième sous-section, nommée **Interthéoricité logicielle**, a permis de présenter le concept d'interthéoricité (Guillaume 2014 ; Guillaume 2020), mettant en exergue une élasticité des théories et permettant une réappropriation et réadaptation de celles-ci à un autre domaine. On s'appuie ainsi sur ce concept d'interthéoricité de Guillaume pour développer le processus que nous nommons interthéoricité logicielle, permettant de rendre rigoureux l'usage de l'interthéoricité dans notre approche du DLT. Celui-ci est composé de deux opérations distinctes : l'exportation originelle et l'importation théorique. La première opération consiste à présenter les définitions originelles des théories, sans toutefois tenir compte des controverses éventuellement liées à celles-ci ; la deuxième à adapter lesdites théories à l'industrie vidéoludique ou au développement de jeux vidéo 3D.

Enfin, nous avons mis en surbrillance, dans la troisième sous-section qui se nomme **Hyperstructure tridimensionnelle**, le concept inédit d'une hyperstructure 3D, c'est-à-dire une structure sémio-informatique de la conception des jeux vidéo en 3D temps réel. Nous avons réussi à développer trois caractéristiques inédites

pour celle-ci : le type d'hyperstructure 3D, interprétatif ou génératif, puis l'interface intemporelle et enfin l'implémentation générationnelle. Une hyperstructure 3D interprétative porte sur l'analyse de la création de jeux vidéo existants, alors qu'une hyperstructure 3D générative va plutôt étudier des outils de développement de jeux. Ces deux aspects sont inspirés de la sémantique textuelle ainsi que de la sémiotique discursive (Couégnas 2022). Enfin, l'interface et l'implémentation sont plutôt liées aux caractéristiques intrinsèques de notre hyperstructure. Autrement dit, l'interface intemporelle porte sur des aspects de l'hyperstructure qui existent potentiellement pour l'ensemble des générations, alors que des implémentations générationnelles sont plutôt liées à des générations précises. Ainsi, l'élaboration puis la définition de notre approche du DLT et de l'hyperstructure 3D permet de pouvoir exposer les analyses diachroniques.

3. CINQUIÈME GÉNÉRATION : PRÉMICES DES MOTEURS 3D

3-1. ASSEMBLAGE DES PREMIERS JEUX

Après avoir évoqué les trois premières générations dans le chapitre d'introduction de la thèse, depuis la *Magnavox Odyssey* jusqu'à la *NES* de Nintendo, nous pouvons succinctement aborder la quatrième génération dans ce texte introductif. Ensuite, et tout au long de ce deuxième chapitre, nous nous concentrerons sur la cinquième génération de consoles. Ainsi, la quatrième génération débuta, comme rappelé par Williams, en « 1987 [lorsque] le fabricant d'ordinateurs NEC et le développeur de logiciels Hudson Soft ont distribué le *PC Engine* au Japon » (Williams 2017, 160 TDR)[217]. Cette même console fut exportée sous une autre appellation, comme le rappelle Scullion, puisqu'elle « est arrivée en Amérique du Nord en 1989 sous le nom de TurboGrafx-16 » (Scullion 2020 TDR)[218]. Néanmoins, la quatrième génération fut principalement marquée par la console *Super Famicom*, distribuée d'après Kent un peu plus tard « au Japon, le 21 novembre 1990 » (Kent 2001, 431 TDR)[219], mais aussi, selon Williams, par la « Sega Mega Drive (...), commercialisée sous le nom "Genesis" aux États-Unis » (Williams 2017, 161 TDR)[220].

Nous allons focaliser notre propos sur la console ayant atteint le plus grand nombre de ventes, autrement dit la *Super Famicom*. Cette machine a également été exportée, d'après Scullion, « en Amérique du Nord le 23 août 1991 avec un (...) changement de nom : la Super Nintendo Entertainment System, ou SNES en abrégé » (Scullion 2020 TDR)[221]. Ainsi, la SNES a comptabilisé, pour reprendre les propos de Sam Byford, près de « 49,1 millions d'unités vendues dans le monde » (Byford 2020 TDR)[222] depuis le début de sa commercialisation, ce qui, selon Levi Buchanan, « dépasse la *Mega Drive*, qui s'est écoulée à (...) 29 millions d'unités vendues » (Buchanan 2016 TDR)[223].

217. « 1987 [when] computer manufacturer NEC and software developer Hudson Soft released the PC Engine in Japan » (Williams 2017, 160).
218. « came to North American in 1989 as the TurboGrafx-16 » (Scullion 2020).
219. « in Japan, in November 21, 1990 » (Kent 2001, 431).
220. « Sega's Mega Drive (...), marketed as the "Genesis" in the United States » (Williams 2017, 161).
221. « in North America on 23 August 1991 with a (...) name change: the Super Nintendo Entertainment System, or SNES for short » (Scullion 2020).
222. « 49.1 million units sold worldwide » (Byford 2020).
223. « surpassing the Genesis, which sold (...) 29 million units. » (Buchanan 2016).

Sur le plan technique et d'après Kent, la SNES possédait un « véritable processeur 16-bit plutôt que l'hybride 8-bit/16-bit de la TurboGrafx » (Kent 2001, 431 TDR) [224], mais son architecture était surtout « conçue (...) autour des performances visuelles et audio plutôt que de la vitesse de traitement » (*ibid.*)[225]. Si le CPU est mis en surbrillance dans ce descriptif, c'est parce qu'il s'agit du composant le plus important dans l'architecture d'un ordinateur. En effet, il est d'usage de mentionner, grâce à Wardyga, que le « *Central Processing Unit* (CPU) est comme le cerveau d'un ordinateur ou d'une console (...) [qui] effectue des calculs et traite les informations indiquant aux autres composants ce qu'il faut faire » (Wardyga 2019, 30 TDR) [226]. On précise qu'il existe plusieurs dénominations de ce composant, d'après Jim Ledin, notamment « (CPU), micro-processeur, ou, simplement, processeur » (Ledin 2022, 54 TDR)[227]. Nonobstant le fait que ces termes semblent interchangeables dans certains contextes, on note avec Kaviyaraj Ravichandran, Mohan Uma et Soumya Ranjan Jena que l'appellation « "processeur", (...) fréquemment utilisée à la place de "CPU" (...) est techniquement incorrecte (...) [puisque] le CPU n'est qu'un type de processeurs parmi d'autres » (Ravichandran, Uma et Jena 2022, 116 TDR)[228]. On peut ainsi souligner l'existence d'autres processeurs spécialisés pour certaines tâches, comme le GPU.

Dans tous les cas, le CPU représente la clef de voûte d'un ordinateur ou d'une console, octroyant de multiples possibilités puisque, selon Justice, il « permet (...) d'exécuter les programmes qui n'ont pas été élaborés au moment où le processeur a été conçu » (Justice 2021, 122 TDR)[229]. Sur le plan historique, nous avons déjà mentionné en introduction la *Fairchild Channel F*, première console de l'histoire à intégrer un CPU et à devenir, selon Kent, la « première console de

224. « true 16-bit processor rather than the 8-bit/16-bit hybrid in TurboGrafx » (Kent 2001, 431).
225. « designed (...) around visual and audio performance rather than processing speed » (Kent 2001, 431).
226. « Central Processing Unit (CPU) is like the brain of a computer or console (...) [that] makes calculations and processes information that tells the other components what to do » (Wardyga 2019, 30).
227. « (CPU), microprocessor, or, simply, processor » (Ledin 2022, 54).
228. « "processor", (...) frequently used in place of "CPU" (...) is technically incorrect (...) [since] the CPU is only one of several processors » (Ravichandran, Uma et Jena 2022, 116).
229. « allows (...) to run programs that weren't even conceived of at the time the processor was designed » (Justice 2021, 122).

salon programmable » (Kent 2001, xii TDR)[230]. Une nouveauté technologique qui, à l'époque, avait modifié les méthodologies de développement de jeux. On peut même évoquer, grâce à Williams, une restructuration complète de l'industrie vidéoludique, en particulier parce que l'usage des « micro-processeurs (...) a transféré les responsabilités de concevoir un jeu depuis les ingénieurs vers les programmeurs informatiques » (Williams 2017, 67 TDR)[231].

Mais si la programmation d'un logiciel exige de communiquer avec un CPU, on pourrait mettre en exergue, et grâce à Justice, une certaine complexité d'une communication directe avec cette partie de la machine, provenant du fait que le « CPU traite la totalité des données en binaire, (...) [et] la plupart des gens ont des difficultés avec (...) les 0 et les 1 » (Justice 2021, 140 TDR)[232]. Pour pallier ces difficultés substantielles, il était possible d'utiliser, pour reprendre les propos de Mike McGrath, le langage « assembleur, (...) également connu sous le nom de code machine symbolique, (...) [qui] est le seul langage de programmation qui parle directement au CPU d'un ordinateur » (McGrath 2021, ch. 1 TDR)[233]. En bref, *l'assembleur* est une transposition directe du binaire, ou encore, selon Justice, une « forme humainement lisible de code machine » (Justice 2021, 144 TDR)[234]. De surcroît, il s'agit surtout d'un type de langage informatique caractérisé, d'après Richard Blum, par des « *mnémoniques*, (...) des mots ressemblant à l'anglais pour représenter des instructions individuelles » (Blum 2005, 10 TDR)[235].

Par ailleurs, *l'assembleur* est plus ancien que les premiers jeux vidéo, puisqu'il fut, d'après Hossein Ashtari, élaboré par « Kathleen Booth, qui a commencé des recherches théoriques sur le concept en 1947 (...) [et en] 1948 (...) le (...) (EDSAC) disposait d'un *assembleur* intégré » (Ashtari 2023 TDR)[236]. En d'autres termes, *l'assembleur* est considéré comme étant le langage le plus bas niveau possible

230. « first programmable home game to use cartridges » (Kent 2001, xii).
231. « microprocessors (...) shifted the responsibilities of designing the game from engineers to computer programmers » (Williams 2017, 67).
232. « CPU always deals with everything in binary, (...) [and] most people have a hard time with (...) 0s and 1s » (Justice 2021, 140).
233. « assembly, (...) also known as symbolic machine code, (...) [which] is the only programming language that speaks directly to the computer's CPU » (McGrath 2021, ch. 1).
234. « human-readable form of machine code » (Justice 2021, 144).
235. « *mnemonics*, (...) English-style words to represent individual instruction codes » (Blum 2005, 10).
236. par « Kathleen Booth, who began theoretical work on the concept in 1947 (...) [and in] 1948 (...) the (...) (EDSAC) had an assembler integrated » (Ashtari 2023).

pour écrire un logiciel, autrement dit le plus proche de la machine après le binaire. Cela étant, *l'assembleur* n'est pas directement compréhensible par un ordinateur, et c'est pour cette raison que, pour reprendre les propos de Scott, la « traduction des mnémoniques en langage machine est devenue le travail d'un programme système connu sous le nom de programme assembleur » (Scott 2016, 6 TDR)[237]. Ainsi, on distingue le *langage assembleur*, permettant d'écrire des instructions pour le CPU, et le *programme* assembleur, un outil dont le rôle est de traduire lesdites instructions en langage machine pour qu'elles deviennent compréhensibles pour un ordinateur.

Néanmoins, précisons qu'un regard néophyte sur la programmation informatique pourrait se méprendre sur *l'assembleur*, imaginant un langage stable et unifié car utilisé pour le développement de jeux vidéo pendant plusieurs générations. À titre illustratif, pendant la troisième génération de consoles, Hugg constate que « presque tous les jeux commerciaux de la NES étaient écrits en langage assembleur 6502 » (Hugg 2019 TDR)[238]. Or, comme le suggère Hugg en attribuant la séquence numérique *6502*, *l'assembleur* représente plutôt une *famille* de langages comprenant plusieurs variantes distinctes. Dans le cas précis mentionné par Hugg, il s'agit donc d'un langage spécialement conçu, comme rappelé par Nathan Altice, pour le « micro-processeur de la NES [qui] exécutait 99% de l'ensemble d'instructions 6502 » (Altice 2015, 29 TDR)[239]. C'est ce point que Altice va ensuite mettre en exergue, expliquant qu'il s'agit d'une « base commune pour une variété de dialectes, chacun spécifique à une famille particulière de CPU » (Altice 2015, 65 TDR)[240]. C'est pourquoi il peut exister des différences, parfois substantielles, entre les multiples variantes disponibles, phénomène accentué par le fait que « le langage assembleur en lui-même n'est pas une grammaire codifiée » (*ibid*. TDR)[241]. Cette spécificité de *l'assembleur* à des conséquences importantes sur le développement logiciel, notamment la nécessité de réécrire un même logiciel d'une manière différente pour chaque type

237. « translating from mnemonics to machine language became the job of a systems program known as an assembler. » (Scott 2016, 6).
238. « almost all NES commercial titles were written in 6502 *assembly* language » (Hugg 2019).
239. « NES microprocessor [which] ran 99% of the 6502 instruction set » (Altice 2015, 29).
240. « common base for a variety of dialects, each specific to a particular CPU family » (Altice 2015, 65).
241. « assembly language in itself is not a codified grammar » (Altice 2015, 65).

de processeur. Naturellement, cette réécriture n'est pas véritablement réalisée en partant de rien, ou *from scratch*. La programmation d'un logiciel sur une première machine permet surtout d'avoir, selon Charles Petzold, « déjà résolu le problème principal d'organisation et d'algorithme » (Petzold 2023, 428 TDR)[242] dans le code original, même s'il faut réécrire ledit logiciel.

En conséquence, chaque console nécessitait l'utilisation d'un langage *assembleur* distinct. La seule exception pourrait être l'utilisation d'un CPU identique à celui d'une autre console, ce qui n'est hypothétiquement pas l'objectif lors de l'élaboration d'une nouvelle console, sauf pour les révisions de consoles existantes. Néanmoins, pour la console *SNES*, Rodrigo Copetti mentionne l'existence de deux langages *assembleur*, où un « programme principal est écrit en pur assembleur 65816 et le pilote musical est écrit en assembleur SPC700 » (Copetti 2023b, TDR)[243]. Par ailleurs, nous tenons à signaler l'existence de forums anglophones sur internet où de nombreux messages indiquent l'utilisation du langage *C* pour le développement de jeux sur SNES (NesDev 2016), un même langage que nous analyserons dans la suite du chapitre. Cette hypothèse pourrait sembler plausible, compte-tenu du fait que de « multiple studios de jeux vidéo (...) ont également développé leur propre équipement interne » (Copetti 2023b, TDR) [244] . Or, à notre connaissance, nous ne possédons aucune source scientifique pouvant directement confirmer, ou infirmer, cette information.

C'est pourquoi nous considérons donc qu'il existe *potentiellement* des jeux développés en totalité ou en partie avec une version du langage *C* sur *SNES*, en plus de *l'assembleur*, sans pour autant pouvoir prouver rigoureusement cette affirmation. En utilisant le fondement de la véridiction technico-discursif, on peut estimer que l'indice de véridiction technique est relativement faible. Certes, pour améliorer cet indice de véridiction, on pourrait mentionner la possibilité de faire de la rétro-ingénierie sur des jeux de la *SNES.* Pour définir cela, nous pouvons affirmer avec Jitender Narula que le « concept derrière la rétro-ingénierie est de déconstruire quelque chose pour comprendre son architecture interne, en vue de

242. « already solved the major organizational and algorithmic problem » (Petzold 2023, 428).
243. « main program is written in plain 65816 assembly and the music driver is written in SPC700 assembly » (Copetti 2023b).
244. « multiple game studios (...) also developed their own in-house equipment » (Copetti 2023b).

créer une copie ou alors dans l'objectif de l'améliorer ou de la modifier » (Narula 2021, ch. 1 TDR)[245].

Dans notre cas de figure, un tel procédé de rétro-ingénierie permettrait de faire des hypothèses sur la manière dont le code source a été écrit. Cependant, ce procédé est inenvisageable pour une thèse scientifique, mais également dans le domaine industriel, car il n'est pas possible de s'assurer de sa légalité. Ainsi, cette interrogation, que nous ne pouvons pas résoudre à l'heure actuelle, revêt tout de même une grande importance, puisqu'il s'agit d'un procédé qui modifie substantiellement la manière de concevoir des jeux.

Au-delà de cette question portant sur les langages de programmation, nous pouvons constater que les outils matériels du développement de jeux vidéo se sont standardisés uniquement à partir de la quatrième génération de consoles. En effet, lors de la troisième génération, le développement de jeux sur la *NES* était plus complexe puisque, d'après Hugg, « il n'y avait pas de kit de développement "officiel", [et] les développeurs tiers bricolaient leurs propres kits, souvent en utilisant une cartouche modifiée reliée à un *PC* via un câble ruban » (Hugg 2019 TDR) [246] . En revanche, on constatait l'apparition de plusieurs kits de développement dès la quatrième génération, notamment pour la *SNES*. Cependant, nous sommes dans l'impossibilité de décrire précisément ces différents kits, principalement à cause d'une absence de sources bibliographiques scientifiques sur ce point. Certes, nous pourrions, par exemple, évoquer l'article *SNES (Super Famicom) Development Kit Hardware* du site *Retro Reversing* (2020b) mais, au regard du fondement de véridiction technico-discursive, nous estimons que l'énonciateur possède ici une expertise non identifiable. De surcroît, les consoles de développement étant par définition inaccessibles pour le grand public, les énoncés de cette source possèdent un indice de véridiction technique très faible.

Dans tous les cas, nous pouvons constater que les scripts de programmation des jeux vidéo dans la quatrième génération possèdent une durée de vie très

245. « concept behind reverse engineering is to break something to understand its internal architecture to build a copy or for the purpose of improvement or modification » (Narula 2021, ch. 1).

246. « there was no "official" development kit, [and] third-party developers hacked together their own, often using a custom cartridge attached via ribbon cable to a PC » (Hugg 2019).

réduite. En effet, ils ne peuvent exister que sur une seule et unique console, et de surcroît dans une seule génération puisqu'ils sont codés dans un langage *assembleur*. Par essence, le code source d'un programme écrit en *assembleur* n'est pas portable, alors même qu'il s'agit d'un aspect fondamental en programmation puisque, comme le rappelle Jonathan Steinhart, un « code qui est portable peut s'exécuter dans un environnement différent de celui pour lequel il a été développé » (Steinhart 2019, 420 TDR)[247]. Assurément, il serait possible de conceptualiser des méthodes pour rédiger un code *assembleur* ayant des parties relativement modulables, mais la portabilité n'est pas envisageable.

Sur ce point d'une modularité d'un code *assembleur*, nous pourrions évoquer les macros et fonctions grâce à Brian R. Hall et Kevin J. Slonka, où les « fonctions sont meilleures pour de longues séquences de code répétable (...) [alors que] les macros sont plus adaptées pour de courtes et simples séquences de code répétable » (Hall et Slonka 2018, ch. 9 TDR)[248]. Cependant, cette modularité reste applicable à une seule famille de processeurs, et ne pourrait résoudre ce problème d'absence de portabilité. Mais il est préférable d'approfondir cette modularité potentielle du code ultérieurement, notamment lorsqu'on effectuera un comparatif avec le langage *C*. Et c'est justement la raison pour laquelle il est désormais temps de naviguer dans les méandres de la cinquième génération de consoles.

247. « code that is portable can run in a different environment that the one for which it was developed » (Steinhart 2019, 420).

248. « functions are best for long sequences of repeatable code (...) [while] macros are best for short and simple sequences of repeatable code » (Hall et Slonka 2018, ch. 9).

3-1-A. Développement primo-générationnel

Dès les années 1990, le début de la cinquième génération a été caractérisé par une métamorphose technologique majeure dans l'industrie vidéoludique, une véritable « avancée dans les graphismes 3D » (Wolf 2008b, 140 TDR)[249], pour reprendre les propos de Wolf. Celle-ci est portée par l'émergence de trois consoles principales : la *Saturn*, la *PlayStation* ainsi que la *Nintendo 64*. Contextuellement, on peut très succinctement mentionner l'accord initial, comme l'explique Kevin Webb, de « Nintendo [qui] s'est associé avec Sony pour développer un nouveau périphérique CD-ROM (...) pour la Super Nintendo » (Webb 2020 TDR)[250]. Un accord qui n'a finalement pas abouti puisque, selon Drew Sleep, « Nintendo a estimé que les termes de l'accord allaient à l'encontre de ses intérêts, et a conclu un accord avec Philips à la place » (Sleep 2022, 84 TDR) [251] . De cet événement résulta la console *PlayStation*, « conçue par l'ingénieur **Ken Kutaragi** (qui a développé la puce sonore de la SNES) » (Wardyga 2019, 200 TDR)[252], et distribuée en dehors d'un quelconque lien avec Nintendo.

Néanmoins, l'architecture d'origine de la *PlayStation* était d'abord centrée sur la 2D, afin de suivre la lignée technologique des consoles précédentes. Cependant, Jamie Lendino indique que le contexte de transformation du marché vidéoludique modifia les perspectives puisque, si « Sony considérait fortement (...) un système en 2D (...), à la suite du succès de Virtua Fighter, Sony (...) redoubla d'efforts pour inclure des (...) graphismes 3D » (Lendino 2020 TDR)[253]. Surtout, parce que Nintendo ciblait encore un public relativement jeune et une technologie assez stable, d'après Kevin Bryan, « Kutaragi (..) était convaincu qu'un véritable système 3D (...) permettrait à Sony de diriger l'ère de la 3D tout comme Nintendo a dirigé l'ère de la 2D » (Bryan 2005, 11 TDR)[254].

249. « leap into 3-D graphics » (Wolf 2008b, 140).
250. « Nintendo [who] partnered with Sony to develop a new CD-ROM peripheral for the (...) Super Nintendo » (Webb 2020).
251. « Nintendo felt the terms of the agreement went against its interests and it struck a deal with Philips instead » (Sleep 2022, 84).
252. « designed by engineer **Ken Kutaragi** (who developed the SNES sound chip) » (Wardyga 2019, 200).
253 « Sony was strongly considering (...) a 2D system (...), in the wake of Virtua Fighter's success, Sony (...) redoubled its efforts to include (...) 3D graphics » (Lendino 2020).
254. « Kutaragi (..) was convinced that a true 3D system, (...) would allow Sony to lead the 3D era just as Nintendo had led the 2D era » (Bryan 2005, 11)

Dans tous les cas, la distribution de cette console débuta précisément, comme rappelé par Kent, à partir du « 3 décembre 1994, [quand] Sony a lancé (...) [la] *PlayStation* sur le marché japonais » (Kent 2001, 502 TDR)[255]. Et, sans retracer un historique complet, nous pouvons affirmer que la *PlayStation* fut la console la plus vendue de la cinquième génération, puisque, comme le rappelle Kent, « Sony allait ensuite vendre plus de 100 millions de PlayStations dans le monde, Nintendo a écoulé près de 33 millions de N64 et Sega a vendu 9,2 millions de Saturn » (Kent 2021, ch. 2 TDR)[256]. Nous pouvons présenter dans le tableau ci-dessous les ventes des consoles de la cinquième génération, en reprenant les informations de Wardyga (2019, 212) et de Kent (2021).

Consoles	Année de distribution	Ventes mondiales approximatives
Panasonic 3DO Interactive Multiplayer	1993	Environ 2 millions
Atari Jaguar	1993	Inférieur à 250 mille
Sega Saturn	1994	Supérieur à 9 millions
Sony PlayStation	1994	Supérieur à 102 millions
Nintendo 64	1996	Supérieur à 32 millions

Tableau 13. Ventes mondiales de la cinquième génération (Wardyga 2019, 212)

Mais, outre son orientation pour la performance dans les jeux en 3D temps réel, celle-ci présentait également des innovations sur les procédés de développement de jeux vidéo. En effet, et Bryan rappelle que ces mêmes innovations sont surtout induites par les « régulations de contenu plus indulgentes par rapport à Nintendo et une bibliothèque de logiciels importante pour faciliter l'apprentissage de la 3D, [qui] ont attiré les développeurs tiers » (Bryan 2005, 13 TDR)[257]. De surcroît, Sony a radicalement changé les

255. « December 3, 1994, [when] Sony launched (...) [the] PlayStation into the Japanese market » (Kent 2001, 502).
256. « Sony would go on to sell more than 100 million PlayStations worldwide, Nintendo sold nearly 33 million N64s, and Sega sold 9.2 million Saturns » (Kent 2021, ch. 2).
257. « lenient content regulations compared to Nintendo and an extensive software library to ease the 3D learning curve, [which] enticed third-party developers » (Bryan 2005, 13).

méthodologies elles-mêmes de l'industrie du jeu vidéo, en permettant de développer des jeux vidéo directement sur un *PC*. C'est en effet ce qui semble avoir été indiqué par le magazine *EDGE UK*, où « le choix le plus ingénieux de la part de Sony a été la décision d'utiliser le PC comme plate-forme de développement » (cité dans Retro Reversing 2020a TDR)[258]. Néanmoins, en utilisant notre concept de véridiction technico-discursive, nous pouvons affirmer que l'indice de véridiction technique est faible.

Toutefois, cette hypothétique décision a eu deux conséquences importantes altérant la création de jeux vidéo sur consoles : rendre le développement plus accessible pour l'ensemble des studios professionnels, mais également permettre un développement simultané sur plusieurs supports. On peut illustrer cela à travers de nombreuses œuvres vidéoludiques, grâce à Patrick Hickey, notamment le « travail sur *Driver* (...) [qui] a commencé sur PC plutôt que sur PlayStation puisque (...) nous n'étions même pas certains que cela serait techniquement possible sur la console PlayStation » (Hickey 2022, 254 TDR)[259]. Et c'est grâce à cette possibilité de développement sur PC que lesdits développeurs de *Driver: You Are the Wheelman* (Atari 1999, PS1) ont eu la possibilité « de passer au développement parallèle du jeu » (*ibid*. TDR)[260].

Néanmoins, cette possibilité de développement multi-plateforme existe grâce au langage *C*, qui modifia substantiellement la manière de concevoir des jeux vidéo. Un même langage existe depuis plusieurs décennies déjà, histoire mise en surbrillance par Peter A. Darnell et Philip E. Margolis, exposant que « le langage C a été développé en 1972 par Dennis M. Ritchie aux AT&T Bell Labs en tant que *langage de programmation système* » (Darnell et Margolis 1996, 5 TDR)[261]. Surtout, le changement principal par rapport à la famille des *assembleurs*, c'est, d'après McShaffry et Graham, que « le langage de programmation C (…) avait la capacité d'être indépendant des plates-formes » (McShaffry et Graham 2013, 331

258. « the most ingenious move on Sony's part was its decision to use the PC as a development platform » (cité dans Retro Reversing 2020a).
259. « work on Driver (...) [which] actually started on PC rather than PS since at that early stage of development we weren't even sure it would be technically feasible on the PlayStation's hardware » (Hickey 2022, 254).
260. « moved to parallel development » (Hickey 2022, 254).
261. « the C language was first developed in 1972 by Dennis M. Ritchie at AT&T Bell Labs as a *systems programming language* » (Darnell et Margolis 1996, 5).

TDR)[262]. C'est pourquoi il était possible de développer un logiciel qui pouvait ensuite fonctionner sur « d'autres ordinateurs en le recompilant sur les machines cibles (...) [un] processus (...) appelé *portage* » (Darnell et Margolis 1996, 5 TDR)[263]. Il s'agissait ainsi d'une nouvelle forme d'abstraction de la machine, avec une indépendance qui rendait, d'après Brian Kernighan et Dennis Ritchie**,** « facile d'écrire des programmes qui s'exécuteront sans modification sur n'importe quelle machine prenant en charge le langage C » (Kernighan et Ritchie 1988, 8 TDR)[264].

De surcroît, on peut mettre en exergue le processus de compilation du langage *C*, octroyant de nombreux avantages pour le développement de jeux vidéo, notamment, d'après Robert W. Sebesta, « une exécution très rapide du programme, une fois que le processus de traduction est terminé » (Sebesta 2016, 48 TDR)[265]. On précise que le *C++*, langage de notre corpus développemental direct et que nous étudierons dans le chapitre ultérieur, est également, comme l'indique le créateur du langage Bjarne Stroustrup lui-même, un « langage compilé (...), son texte source doit être traité par un compilateur, qui produit des fichiers objet, qui sont ensuite combinés par un *linker* produisant un programme exécutable » (Stroustrup 2022 TDR)[266]. Le compilateur doit donc traduire l'ensemble du fichier source pour constituer un code *assembleur*, qui sera ensuite lui-même traduit en langage machine.

Ainsi, l'usage du langage *C*, par ailleurs de plus haut niveau qu'un langage *assembleur*, semblait être inévitable à partir du développement des premiers jeux 3D sur consoles. C'est ce qui peut être également mis en exergue par McShaffry et Graham, puisque « le langage assembleur fonctionnait suffisamment bien pour les anciens jeux, car ils étaient plus petits et simples » (McShaffry et Graham

262. « the C programming language (…) had the capability of being platform independent » (McShaffry et Graham 2013, 331).
263. « other computers by recompiling it on the target machines (...) [a] process (...) called porting » (Darnell et Margolis 1996, 5).
264. easy to write programs that will run without change on any machine that supports C » (Kernighan et Ritchie 1988, 8).
265. « very fast program execution, once the translation process is complete » (Sebesta 2016, 48).
266. « a compiled language (...), its source text has to be processed by a compiler, producing object files, which are combined by a linker yielding an executable program » (Stroustrup 2022).

2013, 331 TDR)[267]. C'est pourquoi nous pouvons former l'hypothèse que de nombreux jeux vidéo étaient développés avec le langage *C* sur la première *PlayStation*. Cela notamment parce que, comme souligné par Ken Horowitz, « les bibliothèques de développement PlayStation étaient en langage C [tandis que] (...) les développeurs de la Saturn devaient apprendre à utiliser les deux micro-processeurs de la machine et à programmer en assembleur » (Horowitz 2016, Sega Saturn Tiger Team TDR)[268]. Et, parce que le CPU de la première *PlayStation* était, selon Copetti, « basé sur les CPU MIPS R3000 » (Copetti 2023a TDR)[269], on pouvait également utiliser *l'assembleur MIPS*.

Dans tous les cas, on peut considérer que la transition d'un langage *assembleur* vers le langage *C* semble motivée par deux raisons principales : un gain de temps substantiel pour le développement et la maintenance d'un logiciel ainsi que l'optimisation automatique de l'exécutable. Ainsi, le gain de temps de développement était principalement lié au fait que, d'après James Feldman et Edward Czeck, la « génération, les tests et la maintenance de grands programmes en code *assembleur* étaient extrêmement coûteuses par rapport à la réalisation de cette même tâche dans un langage de haut niveau » (Feldman et Czeck 2006, ch. 9.1 TDR)[270]. Surtout, on peut mettre en exergue le fait que, selon McShaffry et Graham, « une autre grande différence entre le langage assembleur et le C est que le C est bien plus facile à comprendre pour un être humain » (McShaffry et Graham 2013, 332 TDR)[271]. Cela permettait inévitablement d'augmenter la rapidité d'écriture d'une fonctionnalité lors du développement d'un jeu, même si cela n'implique pas nécessairement une réduction du temps de développement.

267. « Assembly language worked well enough for older games since they were smaller and simpler » (McShaffry et Graham 2013, 331).
268. « the PlayStation's development libraries were in C language [while] (...) Saturn developers had to learn how to use the machine's two processors and program in Assembly » (Horowitz 2016, Sega Saturn Tiger Team).
269. « MIPS R3000-based CPUs » (Copetti 2023a).
270. « generation, testing, and maintenance of large programs in assembly code was extremely expensive when compared to doing the same task in a HLL » (Feldman et Czeck 2006, ch. 9.1).
271. « another huge difference between assembly language and C is that C is much easier for a human to understand » (McShaffry et Graham 2013, 332).

Concernant la deuxième raison, l'optimisation semblait être un argument important pour l'usage du langage *C* puisque, d'après Hyde, « (...) les auteurs de compilateurs optimisés commençaient à affirmer que les performances d'un code généré par un compilateur étaient de l'ordre de 10 à 50 pourcents [plus élevées] que d'un code assembleur optimisé à la main » (Hyde 2020b, 2 TDR)[272]. Cependant, un tel propos de la part des auteurs de compilateurs cachait un certain prérequis presque insidieux, puisque cela supposait que les développeurs avaient déjà une expertise dans le langage *assembleur*. C'est du moins ce que Hyde affirme puisque, « lorsque les programmeurs ont commencé à abandonner le langage assembleur (...) ils comprenaient généralement les implications bas niveau (...) et pouvaient choisir leurs (...) instructions de manière appropriée » (Hyde 2020b, 2 TDR)[273]. En d'autres termes, l'optimisation du compilateur dépend directement des connaissances du langage *assembleur*. Celle-ci est donc nécessaire pour pouvoir augmenter la couche d'abstraction et passer à un langage de plus haut niveau, tout en bénéficiant desdites optimisations techniques.

Toutefois, l'optimisation peut prendre plusieurs formes dans le développement d'un logiciel, en particulier sur des consoles dont la puissance est relativement limitée. Hall et Slonka évoquent le fait que « l'optimisation a généralement deux formes : logicielle (optimisations du compilateur) et matérielle (optimisations du CPU) » (Hall et Slonka 2018, ch. 9 TDR)[274]. Cette optimisation logicielle pourrait donc prendre comme forme l'utilisation du langage *assembleur*. Et, plutôt que d'opposer systématiquement les deux langages, il existait également la possibilité de travailler conjointement en *assembleur* et en *C*. En effet, Blum indique au moins trois moyens de réaliser une telle opération :

272. « optimizing compilers (...) authors began claiming that the performance of compiler-generated code was within 10 to 50 percent [higher] of hand-optimized assembly code » (Hyde 2020b, 2).

273. « when programmers first began giving up assembly language (...) they generally understood the low-level ramifications (...) and could choose their (...) statements appropriately » (Hyde 2020b, 2).

274. « Optimization generally has two forms: software (compiler optimizations) and hardware (CPU optimizations) » (Hall et Slonka 2018, ch. 9).

- ► « Implémenter la fonction *from scratch* en assembleur (...) et l'appeler depuis (...) le C ;
- ► Créer la version en langage assembleur du code en C (...), modifier l'assembleur (...) si nécessaire, puis *linker* le code assembleur pour créer l'exécutable ;
- ► Intégrer le code assembleur (...) dans le code C original et le compiler en utilisant le compilateur C standard » (Blum 2005, 365 TDR)[275].

La troisième option est un procédé spécifique pour certains langages informatiques, qui consiste à mélanger directement *l'assembleur* et un langage de plus haut niveau, en l'occurrence le *C*, afin d'améliorer les performances. Ce phénomène est relativement habituel à l'époque, comme souligné par Stefan Zerbst et Oliver Düvel, puisque « les langages de haut niveau comme le C ou le C++ permettent aux programmeurs d'intégrer du code assembleur (...) cela s'appelle l'assembleur *inline* » (Zerbst et Düvel 2004, 93 TDR)[276].

On peut illustrer l'utilisation de l'assembleur *inline* à travers un exemple hypothétique du jeu *Metal Gear Solid* (1998 Konami, PS1), où il semblerait que « Konami (...) [avait] un programmeur "optimisation" dédié qui passait en revue tout le code pour trouver les éléments cruciaux du moteur à convertir en code assembleur » (Racketboy 2018, TDR)[277]. Néanmoins, en s'appuyant sur notre véridiction technico-discursive, nous ne pouvons pas nous assurer de la véridiction technologique en se basant uniquement sur les propos de Racketboy, puisque la source n'est point scientifique. En revanche, il convient de préciser deux éléments dont nous avons la certitude : l'existence de l'assembleur *inline* lors du développement de jeux sur la première *PlayStation* ainsi qu'une confirmation de l'usage de *l'assembleur* par un document officiel des développeurs. C'est pourquoi nous pouvons valider le critère d'énoncé technologique avéré et vérifiable de la véridiction technico-discursive.

275. « Implement the function from scratch in assembly (...) and call it from (...) C; Create the assembly language version of the C code (...), modify the assembly (...) as necessary, and then link the assembly code to create the executable; Create the assembly (...) code (...) within the original C code and compile it using the standard C compiler » (Blum 2005, 365).

276. « high-level languages such as C/ C++ do enable programmers to embed assembler code right into a function implementation, this is called *inline* assembler » (Zerbst et Düvel 2004, 93).

277. « Konami (...) [had] a dedicated "optimization" programmer that went through all the code to find crucial points of the engine to turn into assembly code » (Racketboy 2018).

Surtout, nous devons obligatoirement évoquer un jeu de sixième génération pour mettre en exergue les nombreux détails présentés dans le documentaire interactif distribué avec le jeu *Metal Gear Solid 2: Substance* (Konami 2002, PS2). Celui-ci, intitulé *The Document of Metal Gear Solid 2* (2003 Konami, PS2), souligne que le moteur de MGS2 est, d'après Konami, « écrit en langage C, [avec] de l'assembleur *inline* utilisé dans certaines parties (...) totalisant plus de 1,15 million de lignes » (Konami 2002, Program System Structure TDR)[278]. Par ailleurs, les développeurs ont également conçu un langage spécifique, le « GCL (Game Command Language) (...) le langage de script conçu pour le développement de MGS » (Konami 2002, Program Script Language TDR)[279]. Toutefois, nous ne pouvons pas affirmer si le GCL a été utilisé pour le premier *Metal Gear Solid* (1998 Konami, PS1).

De surcroît, pour valider l'existence de l'assembleur *inline* à l'époque, nous pouvons nous appuyer sur la standardisation des langages de programmation. En effet, puisque Scott souligne que « presque tous les langages largement utilisés possèdent un standard international officiel ou (...) une seule implémentation canonique » (Scott 2016, 8 TDR)[280]. En ce qui concerne le langage *C*, c'est, d'après Robert Seacord, « en 1983, [que] l'American National Standards Institute (ANSI) a formé le comité X3J11 pour établir une spécification des normes C, et en 1989, la norme C a été ratifiée » (Seacord 2020, xxiv TDR)[281]. C'est pourquoi nous pouvons nous appuyer sur la norme publiée « en 1990 (...) [comme étant] la première édition de la norme C, C90 (ISO/IEC 9899:1990) » (ibid. TDR)[282], antérieure à la publication du premier jeu *Metal Gear Solid* (1998 Konami, PS1). Cette norme *C90* contient le mot-clef *asm*, notamment dans la section *G.5.10* intitulée *The asm keyword*, indiquant que « le mot-clef **asm** peut être

278. « written in C language, [with] inline assembly used in some areas (...) totaling more than 1.15 million lines » (Konami 2002, Program System Structure).
279. « GCL (Game Command Language) (...) the script language designed for the development of MGS » (Konami 2002, Program Script Language).
280. « almost every widely used language has an official international standard or (...) a single canonical implementation » (Scott 2016, 8).
281. « in 1983, [that] the American National Standards Institute (ANSI) formed the X3J11 committee to establish a standard C specification, and in 1989, the C Standard was ratified » (Seacord 2020, xxiv).
282. « in 1990 (...) [as] the first edition of the C Standard, C90 (ISO/IEC 9899:1990) » (Seacord 2020, xxiv).

utilisé pour insérer du code en langage assembleur directement (...) via une instruction de la forme **asm** (chaîne-de-caractère-littérale) ; » (American National Standards Institute 1990, 209 TDR)[283].

En substance, les débuts de l'utilisation du langage *C* dans la création de jeux avaient pour conséquence de créer une différence fondamentale avec les générations précédentes, puisque DeMaria indique que la « technologie 3D a radicalement changé les facettes du *game design* et du développement de jeux » (DeMaria 2019b, 255 TDR)[284]. Désormais, il était nécessaire de repenser intégralement le processus de création, puisque les anciennes techniques devenaient inopérantes pour des jeux technologiques plus complexes. Cette conjoncture va engendrer l'idée de concevoir un moteur de jeu.

283. « the asm keyword may be used to insert assembly-language code directly (...) via a statement of the form asm (character-string-literal) ; » (American National Standards Institute 1990, 209).

284. « 3D technology radically changed the face of game design and development » (DeMaria 2019b, 255).

3-1-B. PROTOTYPAGE DES PREMIERS MOTEURS DE JEU

L'idée d'une déhiscence entre les mécanismes interactifs d'un jeu vidéo précis et des composantes modulaires et réutilisables du *gameplay*, autrement dit ce que nous pourrions plus ou moins nommer un moteur de jeu, semble exister au moins depuis le début des années 1980. Naturellement, au regard des caractéristiques intrinsèques à la famille des langages assembleurs, cette tâche était relativement difficile à accomplir. De notre point de vue, un code réutilisable en assembleur s'apparenterait à un oxymore, puisqu'il ne pourrait avoir aucune portabilité, et celui-ci ne fonctionnerait que pour une seule console. Certes, on pourrait trouver un intérêt pour la conception de plusieurs jeux d'une même console, mais un tel procédé serait indéniablement limité. De surcroît, à cause du caractère confidentiel du processus de développement dans l'industrie vidéoludique, décrit dans des sections précédentes, il est difficile d'esquisser un tableau exhaustif des tentatives de conception de moteurs de jeu.

Toutefois, il est possible de citer plusieurs outils de conception de jeux textuels exclusivement en 2D, que certains auteurs décrivent fondamentalement comme étant des moteurs de jeu. On peut d'abord citer *Adventure Game Interpreter* ou AGI, un potentiel moteur conçu, d'après Raz Greenberg, par « Sierra On-Line (...) [et] développé par l'entreprise pour le premier jeu King's Quest et utilisé pour les jeux d'aventure ultérieurs » (Greenberg 2021, 87 TDR)[285], jusqu'à *Manhunter 2: San Francisco* (Evryware 1989, Multi). Nous pouvons aussi mentionner, grâce à Douglas Schules, Jon Peterson et Martin Picard, le *Gold Box Engine* de l'entreprise *Strategic Simulations Incorporated* puisque, « avec *Pool of Radiance* (1988) SSI a créé un moteur de jeu (nommé le *Gold Box engine*) » (Schules, Peterson et Picard 2018, 118 TDR)[286]. Au regard du développement de jeux sur *NES*, Altice évoque également que les hypothétiques « moteurs de jeu Famicom [qui] étaient beaucoup moins flexibles » (Altice 2015, 126 TDR)[287], illustrant ses propos avec des jeux comme *Super Mario Bros*. (Nintendo

285. « Sierra On-Line (...) [and] developed by the company for the first King's Quest game and used in subsequent adventure » (Greenberg 2021, 87).
286. « with *Pool of Radiance* (1988) SSI created a game engine (referred to as the Gold Box engine) » (Schules, Peterson et Picard 2018, 118).
287. « Famicom game engines [which] were much less flexible » (Altice 2015, 126).

1985, NES). Naturellement, nous éludons une majeure partie des autres supposés moteurs de jeu, qui n'ajouteraient pas d'informations utiles pour notre problématique, puisque nous traitons des jeux en 3D temps réel et non pas en 2D.

Cependant, une interrogation persiste au regard de cette liste d'outils : est-il véritablement question de moteurs de jeu parmi les exemples que nous venons de citer ? Car cette notion d'indépendance entre le jeu et le *gameplay* peut sembler difficile à mettre en place, mais surtout paradoxale dans le contexte technologique de l'époque. Un aspect particulièrement lié à la nature de la famille des langages *assembleurs*, pour lesquels Gregory indique que « les jeux étaient en grande partie codés à la main par les programmeurs (...) [et] la quantité de contenu dans un jeu typique était minuscule » (Gregory 2019, 1024 TDR)[288]. Car la création d'une couche d'abstraction, c'est-à-dire d'un code réutilisable pour les développeurs, pouvait sembler inefficace au regard d'une absence de portabilité, mais aussi de jeux qui ne nécessitaient aucunement cette forme d'abstraction, du moins à l'époque.

De surcroît, les langages *assembleurs* possèdent généralement une relation d'équivalence dite de 1 à 1 avec le code machine ou, d'après la nomenclature sémiotique d'Hébert, une relation formelle dyadique, et plus précisément une « relation **unidirectionnelle**, ou asymétrique, ou non-réciproque » (Hébert 2020, 5 TDR)[289]. Cela signifie qu'une instruction en *assembleur* possède, généralement, une seule traduction en langage machine. Naturellement, il existe des exceptions mais, sur le plan technique, créer un code réutilisable comme pourrait l'être celui d'un moteur de jeu semblerait donc inintéressant, sans pour autant être formellement impossible. De surcroît, si des auteurs considèrent qu'une simple réutilisation, voire une modularisation d'une partie du code dans un jeu vidéo est une occurrence d'un moteur de jeu, alors n'importe quel jeu vidéo pourrait être *de facto* cité comme ayant été développé via un moteur de jeu. Ce qui nous semblerait être une inexactitude sémantique très importante, puisque la

288. « games were largely hard-coded by programmers (...) [and] the amount of content in a typical game was minuscule » (Gregory 2019, 1024).
289. « **unidirectional**, or asymmetrical, or non-reciprocal » (Hébert 2020, 5)

définition d'un moteur n'est pas un simple usage d'un code déjà existant, comme nous le verrons plus loin.

Par ailleurs, c'est la notion d'une standardisation d'outils qui précèdent celle de moteurs de jeu. Et nous pouvons plutôt mettre en surbrillance deux aspects qui vont potentiellement contribuer à l'idée de concevoir cette forme d'abstraction entre l'interactivité d'un jeu précis et des mécanismes interactifs génériques : la standardisation du langage de programmation utilisé, ici le langage *C*, ainsi que celle des API. Et, puisque la première *PlayStation* implique le développement parallèle, c'est-à-dire à la fois sur console et PC, nous pouvons également évoquer des outils de développement PC. Sur ce point, il existerait un consensus sur la genèse de la 3D temps réel, d'après Tomas Akenine-Möller *et al.*, où « beaucoup considèrent l'introduction de la carte [graphique] 3Dfx Voodoo 1 en 1996 comme le véritable début des graphismes tridimensionnels grand public » (Akenine-Möller *et al.* 2008, 1 TDR)[290]. C'est pourquoi nous allons mettre en surbrillance les premières API disponibles sur PC, par exemple *DirectX* et *OpenGL*.

Néanmoins, avant d'aller plus loin, nous pouvons succinctement esquisser la définition d'une API en se référant à Ian Parberry, qui indique que le sigle « API, (...) *application programming interface* (...) est composé d'une collection de bibliothèques [qui peuvent être] compil[ées] dans (...) [un] jeu » (Parberry 2001, 6 TDR)[291]. Et, sur le plan historique, on peut signaler grâce à Peddie la création de la première API « Plot 10 [développé] en 1971 (...) [par] Tektronix (...) pour leurs terminaux graphiques » (Peddie 2022b, 203 TDR)[292]. Dès le début des années 1980, on retrouve ensuite une API majeure, toujours selon Peddie, « IRIS-GL, introduite en 1981 par Silicon Graphics (SGI) (...) [qui] a donné naissance à OpenGL, OpenGL ES et influencé DirectX ainsi que de nombreuses autres APIs

290. « many consider the introduction of the 3Dfx Voodoo 1 [graphics] card in 1996 the real beginning of consumer-level three-dimensional graphics » (Akenine-Möller et *al.* 2008, 1)
291. « API, (...) application programming interface (...) consists of a collection of libraries that (...) [can be] compile[d] into (...) [a] game » (Parberry 2001, 6).
292. « Plot 10 [developed] in 1971 (...) [by] Tektronix (...) for their graphics terminals » (Peddie 2022b, 203).

grand public » (ibid. TDR)[293]. Puis, dès le début des années 1990, nous pouvons surtout mettre l'accent sur les API *DirectX* ou *OpenGL*.

Cela étant, nous ne pouvons pas évoquer les caractéristiques des bibliothèques ou API utilisées pour les multiples consoles de la cinquième génération, puisqu'il s'agit de données confidentielles et inaccessibles. Nous pouvons seulement mentionner le nom des bibliothèques de la première *PlayStation*, grâce à Imagine Media, notamment « les outils de développement Psy-Q (...) utilisés pour le développement de tous les logiciels *PlayStation* existants » (Imagine Media 1996, 80 TDR)[294]. Ce même outil est décrit comme étant un « système de développement de SN Systems et Psygnosis » (*ibid*. TDR)[295]. Néanmoins, cette affirmation provient d'une publicité, et nous ne pouvons garantir la véridiction technologique du propos, au regard de notre fondement de la véridiction technico-discursive.

Quoi qu'il en soit, les API semblaient être particulièrement utiles pour la standardisation des outils de développement de jeux vidéo. Plus précisément, Parberry indique que « DirectX est une API qui permet au programmeur un accès direct au matériel de l'ordinateur plutôt que de passer à travers les nombreuses couches de (...) l'API de Windows » (Parberry 2001, 6 TDR)[296]. En d'autres termes, l'objectif était surtout, selon Peter Kovach, de créer une « couche d'abstraction matérielle (HAL pour *Hardware Abstraction Layer*) qui cache efficacement les dépendances spécifiques du périphérique matériel au développeur » (Kovach 1998, 3 TDR)[297].

Il s'agissait d'un événement majeur dans l'industrie vidéoludique puisque, d'après Allen Sherrod et Wendy Jones, « avant DirectX, les créateurs de jeux vidéo rencontraient des problèmes liés à des incompatibilités matérielles » (Sherrod et

293. « IRIS-GL, introduced in 1981 by Silicon Graphics (SGI) (...) [which] spawned OpenGL, OpenGL ES, and influenced DirectX and many other consumer APIs » (Peddie 2022b, 203).
294. notamment « Psy-Q development tools (...) used for all existing PlayStation software development » (Imagine Media 1996, 80).
295. « development system from SN Systems and Psygnosis » (Imagine Media 1996, 80).
296. « DirectX is an API that allows the programmer direct access to the computer's hardware instead of going through the many layers of (...) Windows API » (Parberry 2001, 6).
297. « hardware abstraction layer (HAL) which effectively hides the device-specific dependencies of the hardware from the developer » (Kovach 1998, 3).

Jones 2012, 1 TDR)[298]. Or, on considère bien, grâce au documentaire Xbox, que c'est « DirectX (...) [qui] a standardisé le développement de jeux sur les différents matériels PC » (Xbox 2021a, 11:13-11:18 TDR)[299], offrant une compatibilité avec les multiples configurations d'ordinateurs. C'est la raison principale pour laquelle nous pouvons attester de l'influence importante des API sur les premiers jeux en 3D, puisque Williams indique que « Direct3D et son principal concurrent, OpenGL, (...) ont permis aux développeurs de jeux de se concentrer sur la création de jeux et ont facilité la transition du PC vers la 3D pendant les années 1990 » (Williams 2017, 201 TDR)[300]. Kovach va également insister sur le système d'exploitation *Windows 95* des ordinateurs qui améliore la standardisation, puisqu'il s'agit de « l'implémentation Windows qui a finalement pris en compte les besoins du développeur de logiciel vidéoludique » (Kovach 1998, 9 TDR)[301]. En l'occurrence, il précise que « DirectX et Windows 95 (...) peuvent désormais facilement prendre en charge n'importe quel choix de matériel de l'utilisateur » (*ibid*. TDR)[302].

En revanche, il existe bien une différence entre *OpenGL* et *DirectX*, mise en exergue par Hawkins *et al*. Il est ainsi indiqué que « OpenGL est simplement une bibliothèque graphique [et] contrairement à DirectX, elle n'inclut pas la prise en charge des (...) aspects qui ne sont pas directement liés aux graphismes » (Hawkins, Astle et Lamothe 2001, 10 TDR)[303]. Ajoutons que certaines sources, notamment Savchenko, mentionnent « la bibliothèque Glide [qui] est une fine couche d'interface pour les accélérateurs graphiques Voodoo » (Savchenko 2000, 321 TDR)[304], qui semblerait être similaire à *OpenGL* dans son principe. Il est ainsi possible de distinguer deux principaux types d'API : une API 2D/ 3D

298. « prior to DirectX, game makers were struggling with problems stemming from hardware incompatibilities » (Sherrod et Jones 2012, 1).
299. « DirectX (...) [which] standardized game development across PC hardware » (Xbox 2021a, 11:13-11:18).
300. « Direct3D and its main competitor, OpenGL, (...) allowed game developers to focus on creating games and facilitated the PC's transition to 3D through the 1990s » (Williams 2017, 201).
301. « the Windows implementation that has finally addressed the needs of the game software developer » (Kovach 1998, 9).
302. « DirectX and Windows 95 (...) can now provide support for any of the user's hardware choices easily » (Kovach 1998, 9).
303. « OpenGL is just a graphics library [and] unlike DirectX, it does not include support for (...) anything else not directly related to graphics » (Hawkins, Astle et Lamothe 2001, 10).
304. « the Glide library [which] is a thin interface layer for Voodoo graphics accelerators » (Savchenko 2000, 321).

comme *OpenGL* ainsi qu'une API Multimédia comme *DirectX*, incluant, comme rappelé par Sherrod et Jones, « les graphismes, le son, le calcul général via le GPU, et les dispositifs d'*input* » (Sherrod et Jones 2012, 2 TDR)[305].

Précisons un dernier point sur l'utilisation des API, qui nécessite ce qu'on appelle l'accélération matérielle. Nous pouvons d'abord définir celle-ci, grâce à Dina Mahmoud, comme étant « l'utilisation d'un matériel spécialisé au sein d'un système informatique conçu pour gérer des tâches spécifiques de manière optimisée » (Mahmoud 2023, 110 TDR)[306]. En l'occurrence, nous utilisons ce terme surtout pour le rendu graphique des jeux vidéo en 3D temps réel et, sur ce point, ce sont surtout les « *Graphics Processing Units* (GPUs) [qui] figurent parmi les accélérateurs matériels les plus célèbres conçus pour le rendu graphique » (*ibid*. TDR)[307]. De surcroît, sur le plan matériel, nous pouvons affirmer, grâce à Peddie, que « la cinquième génération de consoles de jeux (...) a introduit l'accélération matérielle pour les graphismes en 3D sur le marché grand public » (Peddie 2022a, 235 TDR)[308]. Mais, dans le cas de l'absence de celle-ci, et comme le souligne David Eberly, autrement dit lorsque « il n'existe pas d'accélération matérielle ou qu'une API standard est indisponible, le (...) système graphique entier (...) fonctionne sur le CPU (...) [ce qui] est appelé un rendu logiciel » (Eberly 2005, 149 TDR)[309] ou *Software Renderer* en anglais. Dans tous les cas, nous estimons qu'il est intéressant de synthétiser les dates de publication des différentes API dans le tableau ci-dessous, en s'appuyant sur les années indiquées par Peddie (2022b, 203-215). Naturellement, il existait d'autres API à l'époque. Cependant, notre objectif n'est pas de fournir une liste exhaustive, mais seulement d'identifier les API graphiques les plus influentes.

305. « graphics, sound, GPU general purpose computing, and input devices » (Sherrod et Jones 2012, 2).
306. « the use of specialized hardware within a computing system designed to handle specific tasks in an optimized way » (Mahmoud 2023, 110).
307. « Graphics Processing Units (GPUs) [which] are among the most famous hardware accelerators designed for rendering graphics » (Mahmoud 2023, 110).
308. « the fifth generation of game consoles (...) brought 3D accelerated graphics hardware to the home market » (Peddie 2022a, 235).
309. « no hardware acceleration exists or a standard API is unavailable, the (...) entire graphics system (...) run on the CPU (...) [which] is called a software renderer » (Eberly 2005, 149).

API Graphique	Année de Publication	Type d'API
Tektronix Plot 10	1971	API 2D
IRIS-GL 1.0	1983	API 2D et 3D
OpenGL 1.0	1992	API 2D et 3D
DirectX 1.0	1995	API Multimédia

Tableau 14. Genèse des API graphiques principales (Peddie 2022b, 203-215)

Ainsi, on constate que les API représentaient une couche d'abstraction technologique pour les graphismes en 3D temps réel. Or, le moteur de jeu va plutôt constituer un ensemble d'outils spécialisés dans la conception de jeux vidéo, intégrant *de facto* le rendu graphique comme une composante essentielle du moteur. Et, sur ce point, comme souligné par Henry Lowood, on évoque souvent « la "Création du moteur de jeu" (...) [avec] John Carmack, le lead programmeur chez id » (Lowood 2014, 181 TDR)[310]. C'est pourquoi on peut indiquer, grâce à Thomas Haigh et Paul Ceruzzi, le fait que « DOOM a introduit le concept du moteur de jeu, en séparant le code nécessaire pour gérer les événements dans le monde du jeu (...) depuis les "*assets*" comme des objets [et] des monstres » (Haigh et Ceruzzi 2021, 322 TDR)[311]. Ou, en d'autres termes avec Lowood, on peut présenter cela comme en « séparant l'exécution des fonctionnalités principales par le moteur de jeu des *assets* créatives » (Lowood 2014, 181 TDR)[312]. L'un des objectifs principaux de ce moteur de jeu était également de pouvoir générer, selon John Romero, de « nouvelles sources de revenus [et] (...) en vendre des licences » (Romero 2023, ch. 10 TDR)[313].

Néanmoins, l'expression figée *moteur de jeu*, d'après Lowood, « a fait sa première apparition dans des publications après le lancement de DOOM [et] était

310. « the "Invention of the Game Engine" (...) [with] John Carmack, the lead programmer at id » (Lowood 2014, 181).
311. « DOOM introduced the concept of the game engine, by separating the code needed to manage events in the game world (...) from the "assets" such as objects [and] monsters » (Haigh et Ceruzzi 2021, 322).
312. « organizing the components of computer games by separating execution of core functionality by the game engine from the creative assets » (Lowood 2014, 181).
313. « new revenue sources [and] (...) license it » (Romero 2023, ch. 10).

fréquemment mentionnée pendant l'année 1994 » (Lowood 2014, 182 TDR)[314]. Surtout, Romero et Carmack, les cocréateurs de *DOOM* (id Software 1993, Multi), ont adopté ce terme dès la création de *Commander Keen in Keen Dreams* (id Software 1991, Multi) puisque « Carmack et moi [NDT : Romero] avons créé et partagé du code que nous avons commencé à appeler le Keen Engine [et] à ce moment-là, je ne me souviens pas avoir déjà entendu quelqu'un parler d'un moteur de jeu » (Romero 2023, ch. 10 TDR)[315]. Il semblerait également que cette appellation soit inspirée du milieu automobile, puisque « nous [NDT : Carmack et Romero] adorions tous les deux les voitures, et les moteurs donnent aux véhicules la puissance nécessaire pour tout faire fonctionner » (*ibid*. TDR)[316]. C'est pourquoi, en nous appuyant à la fois sur les travaux de Lowood et les propos de Romero, nous considérons que la qualification de *moteur de jeu* pour des outils antérieurs relève d'un pur anachronisme. Une exception pourrait être, en reprenant les informations de Sleep, celle du « moteur Freescape, qui permettait aux joueurs de naviguer dans des environnements 3D depuis une perspective à la première personne » (Sleep 2022, 76 TDR)[317], avec pour premier jeu publié *Driller* (Incentive Software 1987, Multi). Néanmoins, il n'y a aucune mention de la mise à disposition d'une licence pour l'utilisation du moteur *Freescape* par d'autres développeurs.

Pour autant, et comme l'indique Romero, le *Doom Engine* n'affichait pas réellement des graphismes en 3D temps réel, et « avec *DOOM* et *DOOM II*, la plupart des joueurs croyaient qu'ils évoluaient dans un véritable monde en 3D, mais il s'agissait d'une illusion soigneusement élaborée » (Romero 2023, ch. 15 TDR)[318]. C'est pourquoi Romero et Carmack évoquaient une technologie en « 2.5D, et bien que cela soit véritablement impressionnant, cela limitait le

314. « first appeared in print during the year after DOOM's launch [and] occurs frequently during 1994 » (Lowood 2014, 182).
315. « Carmack and I [TN: Romero] created and shared code that we began calling the Keen engine [and] at that time, I don't recall ever having heard someone refer to a game engine » (Romero 2023, ch. 10).
316. « we [TN: Carmack and Romero] both loved cars, and engines are what give vehicles the power to make everything run » (Romero 2023, ch. 10).
317. « Freescape engine, which allowed players to navigate 3D environments from a first-person perspective » (Sleep 2022, 76).
318. « With *DOOM* and *DOOM II*, most players believed that they were playing in a true 3D world, but it was a carefully crafted illusion » (Romero 2023, ch. 15).

potentiel de conception » (*ibid*. TDR)[319]. Néanmoins, avec le développement du jeu *DOOM* (id Software 1993, Multi) et de son moteur de jeu, Romero a également indiqué avoir « commencé [le développement de] (...) DoomEd, l'éditeur de niveaux (...) [qui] nous permettait de construire une carte de (...) chaque niveau et tous les objets qui y apparaissaient » (*ibid*., ch. 12 TDR)[320]. Le concept d'un éditeur de niveaux va se révéler important pour de nombreux moteurs de jeu, comme nous le verrons dans les chapitres ultérieurs.

Cependant, la notion même de moteur de jeu était encore indubitablement associée au jeu *DOOM* (id Software 1993, Multi) puisque, comme le rappelle Mike Thomsen, « au milieu des années 1990, les "moteurs" étaient souvent utilisés pour créer des clones, ou ce qui était autrefois nommé les *DOOM-like*, c'est-à-dire les FPS » (Thomsen 2012 TDR)[321]. Cela implique que l'idée d'un moteur de jeu, du moins lors de cette période, ne semblait pas réellement être un outil indépendant des *assets* du jeu, et encore moins du genre vidéoludique FPS. Un moteur était, en effet, encore trop associé au jeu *DOOM* (id Software 1993, Multi) ou à d'autres jeux similaires sur le plan du *gameplay*. C'est également le cas du *Build Engine*, d'après Williams, un autre moteur de jeu « conçu par Ken Silverman, [et qui] était capable de concevoir des niveaux complexes et des effets spéciaux qui surpassaient le *DOOM engine* mais qui s'appuyait toujours sur des *sprites* 2D dans un espace 3D » (Williams 2017, 184 TDR)[322].

Cette association entre le genre FPS et le concept du moteur de jeu a perduré pour Romero et Carmack. En effet, après la conception du *DOOM Engine*, leur objectif était ensuite de développer le *Quake Engine*. Néanmoins, celui-ci va avoir pour but de dépasser les limitations technologiques du *DOOM Engine* puisque, « en 1995, (...) si nous voulions un moteur qui soit rapide, entièrement en 3D, texturé, (...) et capable de créer Quake, nous devions l'inventer » (Romero 2023,

319. « 2.5D, and while it was genuinely amazing, it limited the design potential » (Romero 2023, ch. 15).
320. « started [the development of] (...) DoomEd, the level editor (...) [that] allowed us to construct a map of (...) each level and all the objects that appeared there » (Romero 2023, ch. 12).
321. « In the mid-90's "engines" were often used to make clones, or DOOM-likes as the FPS was once called » (Thomsen 2012).
322. « designed by Ken Silverman, [and which] was capable of complex level design and special effects that surpassed the DOOM engine but still relied on 2D sprites in a 3D space » (Williams 2017, 184).

ch. 15 TDR)[323]. Et, Gregory constate, bien plus tard et grâce au « code source de Quake et Quake II (...) disponible librement, (...) [que] l'architecture des Quake Engines originaux est raisonnablement bien conçue et "propre" (bien que (...) dépassée et entièrement écrite en C) » (Gregory 2019, 32 TDR)[324].

Toutefois, nous éludons une analyse complète du *Quake Engine* ainsi que de son successeur, *Quake II Engine*. Nous souhaitons néanmoins mentionner deux aspects particuliers de la première version dudit moteur : une technologie 3D nouvelle ainsi qu'un langage de script conçu spécifiquement pour le moteur. Ainsi, on peut mettre en surbrillance, avec Eddie Lohmeyer, sa « capacité à faire un rendu d'un monde en 3D temps réel qui repose sur deux techniques graphiques : la partition binaire de l'espace (BSP) et l'éclairage pré-calculé » (Lohmeyer 2023, 99 TDR)[325]. Et, concernant le langage spécifiquement élaboré pour le *Quake Engine*, on peut préciser, toujours avec Lohmeyer, que « Carmack a développé le langage de programmation unique QuakeC en 1996 pour permettre la modification et l'extension du moteur » (Lohmeyer 2023, 98 TDR)[326].

Par ailleurs, sans entrer dans des détails sémantiques, ajoutons que le moteur parfois nommé *id Tech 1* correspond, comme rappelé par Gavin Annand, au *«* Doom engine, comme il était alors appelé » (Annand 2023 TDR)[327]. Ce même moteur hérite également des anciens moteurs mentionnés par Romero (2023) comme le *Keen Engine*, le *Catacomb 3-D Engine* ou encore le *Wolfenstein 3D Engine*. Notons sur ce dernier point que Romero n'utilise jamais le nom *id Tech* ou *id Tech Engine* dans son ouvrage de 2023.

Au-delà des différents moteurs de l'entreprise *id Software*, on peut surtout mettre en exergue le développement du premier moteur *Unreal Engine* pour le jeu éponyme, *Unreal* (Epic Games 1998, PC). Ce même moteur a été développé, comme indiqué par Chris Plante, par « Sweeney [qui] a commencé (...) en 1995

323. « in 1995, (...) if we wanted an engine that was high-speed, full-3D, texture-mapped, (...) and capable of making Quake, we had to invent it » (Romero 2023, ch. 15).
324. « Quake and Quake II source code (...) freely available, (...) [that] the original Quake engines are reasonably well architected and "clean" (although (...) outdated and written entirely in C) » (Gregory 2019, 32).
325. « ability to render a real-time 3D world is predicated on two graphics techniques: binary space partitioning (BSP) and precalculated lightning » (Lohmeyer 2023, 99).
326. « Carmack developed the unique programming language QuakeC in 1996 to allow for modification and expandability to the engine » (Lohmeyer 2023, 98).
327. *«* Doom engine, as it was then known » (Annand 2023).

(...) [et] en 1998, le jeu et le moteur étaient toujours inachevés » (Plante 2012 TDR)[328]. Un moteur qui semble avoir été, d'après Romero, « clairement influencé par le design et la technologie de *Quake* » (Romero 2023, ch. 19 TDR)[329], y compris pour l'association entre le genre vidéoludique FPS et le moteur de jeu. En effet, le jeu *Unreal* (Epic Games 1998, PC) est un FPS.

Parmi les caractéristiques importantes du moteur, on retrouve « le langage de programmation, *UnrealScript*, ainsi que l'éditeur de niveaux, UnrealEd » (Plante 2012 TDR)[330], qui semble être inspiré du *DoomEd*. Néanmoins, Plante indique que, dans la même année, « le 22 mai 1998, *Unreal* a été distribué (...) [et a eu] des ventes considérables, ainsi qu'une communauté fidèle qui a pleinement utilisé Unreal script » (Plante 2012 TDR)[331]. On précise également que ce moteur a été programmé en *C++*.

Nous pouvons cependant estimer que le moteur *Unreal Engine* va subrepticement s'éloigner de cette tendance à associer moteur de jeu et FPS avec l'exemple *Harry Potter and the Sorcerer's Stone* (2001 KnowWonder, PC), un jeu d'action aventure. D'après Ivan Sulic, la version PC a bel et bien été « créée via le moteur d'Unreal Tournament, [et] le jeu (...) présente ce style classique distinctif propre à UT, avec des châteaux et des couleurs » (Sulic 2012 TDR)[332]. On peut en effet souligner, grâce à Brandon Reinhart, que « l'architecture fortement orientée objet du moteur d'Unreal Tournament le rend extrêmement modulaire » (Reinhart 2000 TDR)[333]. Ainsi, cette même « modularité a permis (...) aux programmeurs d'effectuer des modifications massives à certaines parties du jeu sans affecter d'autres fonctionnalités » (*ibid*. TDR)[334].

Nous pouvons résumer la conception des multiples moteurs de jeu dans le tableau ci-dessous, et mentionner quelques jeux notables pour chaque moteur.

328. « Sweeney [who] began (...) in 1995 (...) [and] by 1998, the game and the engine were still unfinished » (Plante 2012).
329. « clearly influenced by the design and tech of *Quake* » (Romero 2023, ch. 19).
330. « the coding language, UnrealScript, along with a map editor, UnrealEd » (Plante 2012).
331. « on may 22, 1998, *Unreal* shipped (...) [and had] considerable sales, and a loyal community that took advantage of the Unreal script » (Plante 2012).
332. « built off the Unreal Tournament engine, [and] the game (...) portrays that classic distinct glossy UT look, all castles and color » (IGN Staff 2012).
333. « Unreal Tournament engine's strong object-oriented design makes it extremely modular » (Reinhart 2000).
334. « modularity allowed (...) programmers to make massive changes to parts of the game without affecting other features » (Reinhart 2000).

Moteur de jeu	Type	Année	Jeux notables
Freescape	**3D**	1987-1992	*Driller, Castle Master*
Keen Engine	2.5D	1991	*Commander Keen in Keen Dreams*
Catacomb 3-D Engine	2.5D	1991	*Catacomb 3-D*
Wolfenstein Engine	2.5D	1992	*Wolfenstein 3D*
Doom Engine	2.5D	1993-1996	*DOOM, DOOM II*
Build Engine	2.5D	1995-1999	*Witchaven, Duke Nukem 3D*
Quake Engine	**3D**	1996-2002	*Quake*
Quake II Engine	**3D**	1997-2001	*Quake II*
Unreal Engine 1	**3D**	1998-2003	*Unreal, Harry Potter and the Sorcerer's Stone (PC)*

Tableau 15. Genèse des moteurs de jeu en 2.5D et 3D

Dans tous les cas, les multiples moteurs de jeu cités jusqu'à présent étaient conçus pour un usage professionnel, conformément à notre approche du DLT. Cependant, la distinction entre un développement dit amateur et professionnel aura certaines répercussions à long terme sur l'industrie vidéoludique. Un aspect que nous souhaitons particulièrement mettre en relief dans les chapitres ultérieurs, mais également dans la prochaine sous-section.

3-1-C. Obfuscation du développement amateur

Au regard des outils mentionnés, on constate que le développement de jeux vidéo lors de la cinquième génération semblait uniquement accessible aux studios professionnels. Ainsi, l'industrie vidéoludique semblait être un milieu relativement fermé, et nous sommes d'accord avec les propos de John Szczepaniak lorsqu'il indique que, « au milieu et à la fin des années 1990, (...) le développement sur console était extrêmement contrôlé, nécessitant des accords de licences et des kits de développement onéreux » (Szczepaniak 2012 TDR)[335]. C'est pourquoi nous formulons une première distinction entre le développement de deux types de jeux vidéo : les jeux vidéo amateurs et les jeux vidéo commerciaux. Naturellement, l'acception du terme *amateur*, ici, n'est en rien liée à des considérations péjoratives, ni même sur le fait de porter un quelconque jugement sur la qualité potentielle du travail. En effet, nous utilisons surtout l'acception d'amateur désignant une « personne qui pratique un art ou un sport sans en faire profession ni en tirer profit. » (Académie française 2024). Celle-ci est donc seulement opposée au développement d'un jeu vidéo rémunéré ou ayant un objectif commercial, qu'on qualifierait *in fine* de jeu vidéo commercial ou commercialisé. La traduction anglaise *hobbyist* serait potentiellement moins connotée et plus proche de l'acception que nous employons dans cette thèse.

Au-delà de ces considérations linguistiques, nous pouvons évoquer l'existence d'outils spécifiques pour le développement amateur de jeux vidéo dès la cinquième génération de consoles. Ainsi, en portant notre attention uniquement sur la première *PlayStation*, nous pouvons signaler la distribution d'une console spécifique qui rendait le développement de jeux vidéo accessible pour le grand publique : la *Net Yaroze*. Cette variante spécifique de la *PlayStation* a été, selon Bryan, « distribuée en 1997 (...) [et] permettait aux utilisateurs d'écrire leurs propres jeux vidéo pour la plate-forme de Sony » (Bryan 2005, 18 TDR)[336]. Pour la première fois, le grand public semblait posséder une véritable console de développement pour réaliser des jeux, mais Dan Chamberlain souligne un

335. « in the mid-to-late 1990s, (...) console development was tightly controlled, requiring licensing agreements and expensive development kits » (Szczepaniak 2012).
336. « released in 1997 (...) [and] allowed users to write their own videogame for the Sony platform » (Bryan 2005, 18).

« facteur qui a freiné l'adoption plus large de la *Net Yaroze* (...) le prix d'achat (...) aux alentours de 750 dollars » (Chamberlain 2015 TDR)[337]. Pour autant, si cette console était bel et bien, comme l'indique Dimitris, un « kit de développement (...) elle comportait quelques limitations par rapport aux kits de développement PS1 traditionnels » (Dimitris 2020, 8:50-8:57 TDR)[338].

En effet, la restriction principale qui différenciait cette console du développement professionnel était le fait, toujours d'après Dimitris, « qu'il n'y avait aucun logiciel conçu pour une version jouable (...) pour jouer à un jeu *Net Yaroze* de quelqu'un d'autre, vous aviez besoin du kit entier de développement *Net Yaroze* » (Dimitris 2020, 11:28-11:40 TDR)[339]. L'absence de distribution d'un jeu conçu sur une version Net Yaroze signifiait *de facto* une impossibilité d'un véritable développement commercial.

Naturellement, il existait des possibilités de pouvoir exporter un jeu conçu avec une *Net Yaroze*, mais un tel procédé semblait relativement complexe. C'est pourquoi, malgré une potentielle difficulté, quelques jeux émergèrent, notamment *Blitter Boy – Operation: Monster Mall* (Chadwick 1998, PS1) ou encore *Terra Incognita* (Team Fatal 1997, PS1), même s'il ne s'agit pas de jeux réellement en 3D. Pour autant, si la *Net Yaroze* permettait la création de jeux en 3D, l'absence de performance rendait les choses plus complexes.

Néanmoins, nous pouvons également souligner que la *Net Yaroze* permettait l'usage d'une technologie similaire au développement de jeux vidéo commerciaux, notamment au regard du langage de programmation. En effet, puisque le « développement sur la Net Yaroze exigeait (...) d'être familier avec le C [et], à l'époque, le C++ n'était pas une option » (Dimitris 2020, 5:52-6:00 TDR)[340]. Nous éludons pour le moment *le C++,* que nous évoquerons seulement à partir de la sixième génération. Néanmoins, nous ne trouvons en effet aucun document attestant de l'utilisation du *C++* pour le développement d'un jeu sur la première

337. « factor that dampened the more widespread adoption of the Net Yaroze (...) the cost of purchase (...) around $750 » (Chamberlain 2015).
338. « dev kit (...) it came with some limitations over traditional PS1 dev kits » (Dimitris 2020, 8:50-8:57).
339. « there was no player software designed for it (...) to play someone else's Net Yaroze game you would need the entire Net Yaroze development setup » (Dimitris 2020, 11:28-11:40).
340. « development on Net Yaroze required (...) to be familiar with C [and], at the time, C++ wasn't an option » (Dimitris 2020, 5:52-6:00).

PlayStation, aussi bien pour le développement professionnel qu'amateur. Pour autant, il était nécessaire de programmer un jeu intégralement *from scratch* puisque, selon Chamberlain, « les jeux devaient être codés à partir de rien en C, bien que les utilisateurs aient eu accès à certaines bibliothèques spécifiques à la Yaroze » (Chamberlain 2015 TDR)[341].

Dans tous les cas, cette utilisation d'un langage réservé aux professionnels contrastait avec d'autres systèmes de développement de jeux vidéo amateurs sur les consoles de la même génération ou de générations antérieures. Par exemple, Robin Wilde mentionne la *Game Basic for Sega Saturn*, où il indique que « les adeptes de Game Basic créaient des démos 3D assez rudimentaires au milieu des années » (Wilde 2023, 61 TDR)[342]. Concernant la *Nintendo 64*, il n'existe aucun outil de conception de jeux similaire au *Game Basic for Sega Saturn*, à notre connaissance, et « programmer des jeux requiert l'utilisation du kit de développement officiel et une solide connaissance du C – loin d'être le plus simple pour des développeurs professionnels, et encore moins pour des amateurs enthousiastes » (*ibid.*, 60 TDR)[343].

Sur ordinateur, il existait des outils pour le développement de jeux vidéo amateur, notamment un logiciel qui s'appuie sur le moteur *Freescape*. Nommé *Virtual Reality Studio* ou *3D Construction Kit* en dehors des États-Unis selon Travis Fahs (2012), ce kit était un « ensemble complet d'outils de développement pour que tous les utilisateurs puissent essayer de concevoir un jeu Freescape à la maison » (Fahs 2012 TDR)[344]. Or, nous souhaitons plutôt éluder ces outils, ainsi que de nombreux autres logiciels de création qui ne permettent pas de développer des jeux vidéo en 3D temps réel. De surcroît, et au regard des différents ouvrages consultés, nous émettons l'hypothèse qu'il était surtout d'usage de développer un jeu vidéo amateur en créant son propre moteur de jeu. Cela se faisait en utilisant une, voire plusieurs API comme celles mentionnées

341. « games had to be coded from scratch in C, though the users had access to certain Yaroze-specific libraries » (Chamberlain 2015).
342. « Game Basic folks [were] creating fairly rudimentary 3D demos in the mid-2000's » (Wilde 2023, 61).
343. « programming games requires using the official development kit and a strong knowledge of C – not exactly the easiest prospect for professional developers, let alone enthusiastic amateurs » (Wilde 2023, 60).
344. « complete set of development tools so that all the users at home could try their hand at designing a Freescape game » (Fahs 2012).

précédemment, notamment *OpenGL* ou *DirectX*. Néanmoins, nous ne pouvons pas trouver des sources scientifiques pouvant clairement établir un tel constat, et considérons ainsi qu'il s'agit d'une simple hypothèse.

En substance, nous pouvons mettre en exergue une tentative de Sony pour démocratiser le développement de jeux vidéo amateurs sur consoles à travers la distribution de la *Net Yaroze*. De surcroît, cette méthode de développement d'un jeu vidéo utilise le même langage de programmation que celle du développement professionnel, autrement dit le langage *C*. Nous remarquons ainsi qu'il s'agit d'une première tentative de convergence entre les outils de développement de jeux vidéo amateurs et professionnels. Désormais, et afin d'analyser de manière exhaustive et précise les phénomènes observés jusqu'à présent, il est nécessaire de développer une sémiotisation des univers 3D.

3-2. SÉMIOTISATION INTERACTIVE DES UNIVERS 3D

Les informations principales sur la genèse des premiers moteurs de jeu en 3D temps réel nous permettent désormais de répondre de manière plus efficace au besoin primordial de notre recherche : définir l'unité sémio-informatique fondamentale de l'interactivité. Une définition d'une importance cruciale, octroyant la possibilité d'identifier rigoureusement les multiples unités qui représentent des mécanismes interactifs. Nous vérifierons également si les unités en question sont éventuellement limitées par les genres vidéoludiques. De surcroît, notre approche du DLT implique la production de théories sémio-informatiques portant sur les signes, puisque nous estimons, grâce à Gilles Thérien, que la « théorie des signes (...) devrait être au centre des propos de la réflexion sémiotique » (Thérien 1989, 75).

Cependant, la conception de signes sur l'interactivité nécessite d'abord de fournir des précisions importantes sur notre définition d'un mécanisme interactif. Illustrée par Rouse III, cette même définition implique dans notre cas un « *gameplay* [qui] est (...) l'interactivité que le jeu inclut, c'est-à-dire comment les joueurs peuvent interagir avec le monde virtuel et comment ce même monde réagit aux choix des joueurs » (Rouse III 2005, XX TDR)[345]. Nous procédons cependant à une légère modification de celle-ci, en remplaçant le terme *joueur* par *entité*, puisque notre approche du DLT ne porte aucunement sur les joueurs. De plus, nous substituons également le terme *interagir* par *action*, que nous définissons par rapport au dictionnaire de l'Académie française, où il s'agit d'une « opération d'un agent quelconque (...) [et] le même substantif exprimant en général l'opération, le processus et le résultat de l'action considérée » (Académie française 2024). Ainsi, nous pouvons synthétiser notre définition dans le tableau ci-dessous.

345. « gameplay [which] is the (...) interactivity that the game includes, i.e., how players are able to interact with the game-world and how that game-world reacts to the choices players make. » (Rouse III 2005, XX).

Un mécanisme interactif désigne une action quelconque réalisée par une entité, entraînant une modification dans le monde virtuel d'un jeu vidéo en 3D temps réel.

Tableau 16. Définition de la notion de mécanisme interactif

Naturellement, notre objectif n'est pas d'entrer dans des débats sémantiques sur la définition même de l'interactivité, particulièrement parce que, comme le rappelle Lori Landay, « la définition de l'interactivité a été historiquement contestée, avec des universitaires de différents domaines portant leur attention sur la technologie, le cadre de communication ou encore le récepteur » (Landay 2014, 173 TDR)[346]. C'est pourquoi la définition de Rouse III (2005) semble suffisante au regard de nos objectifs ainsi que de notre problématique, traitant surtout de l'existence de limitations technologiques. Et, après avoir esquissé cette définition, nous sommes maintenant en mesure de concevoir deux unités sémio-informatiques élémentaires inédites, que nous allons définir dans cette sous-section en nous appuyant sur le concept fondamental du signe linguistique :

- ► Le signe virtuel interactif haut niveau, ou plus succinctement, le signe interactif ;
- ► Le signe (virtuel) interactif bas niveau.

En revanche, pour renforcer la compatibilité avec notre premier principe immuable de la passerelle technologique infrangible, nous décidons également d'utiliser le terme *gameplay* comme représentant la somme de l'ensemble des signes interactifs. De surcroît, pour élaborer ce signe interactif, nous allons principalement nous appuyer sur des concepts développés dans le CLG de Pseudo-Saussure, pour reprendre l'appellation de Bouquet (2012). Cela permettra de créer une version haut niveau du signe interactif. Par la suite, nous allons également nous inspirer de la subdivision opérée par Hjelmslev pour élaborer une variation bas niveau de ce même signe, en évitant toutefois de reprendre sa terminologie, que nous jugeons moins pertinente dans notre

346. « the definition of interactivity has historically been contested, with scholars from different fields emphasizing either technology, the communication setting, or the perceiver » (Landay 2014, 173).

contexte. On précise également que notre distinction entre haut niveau et bas niveau découle naturellement de l'informatique, c'est pourquoi le signe bas niveau est davantage associé à l'aspect technique, alors que le signe haut niveau représente plutôt un aspect purement conceptuel.

Désormais, nous pouvons véritablement présenter le signe interactif, notre premier concept fondamental inédit qui s'inspire du signe linguistique saussurien¸ représentant pour Stéphane Pagès une « approche dyadique qui fait du signe une entité à deux faces, un contenu et un contenant, dont (...) Saussure [est] à l'origine des néologismes (...) : signifié/ signifiant » (Pagès 2019, 98). À ce titre, nous pouvons précisément citer la définition du CLG, où il est possible de considérer que « le signe linguistique est (...) une entité psychique à deux faces » (Saussure 2005, 74) qui « unit non une chose et un nom, mais un concept et une image acoustique » (*ibid.*, 73). Nous devons également mentionner le référent, même si Hanania constate que le CLG « semble exclure tout renvoi à un référent » (Hanania 2010, 179). Or, nous allons bien inclure celui-ci dans certaines analyses, en nous appuyant sur « l'approche sémiotique (...) [où] le mot sert à désigner une entité du monde appelée **référent** par l'intermédiaire d'un **concept** » (Zufferey et Moeschler 2021, Ch. 10).

Nous élaborons donc notre définition du signe interactif en nous appuyant sur ces trois concepts, que nous adaptons afin qu'ils répondent aux exigences de notre approche du DLT. Ainsi, le signifiant virtuel représente au moins une action permettant de déclencher un mécanisme interactif, voire de le réaliser dans son intégralité. Et le signifié virtuel représente le changement d'état conceptuel dans le monde virtuel produit par cette action. Le référent virtuel, quant à lui, fait plutôt référence à la modification réelle dans ce même monde virtuel. Désormais, nous pouvons synthétiser la définition du signe interactif haut niveau dans le tableau ci-dessous.

Signifiant virtuel	Signifié virtuel	Référent virtuel
Action(s) pour déclencher un mécanisme interactif	Résultat ou changement d'état engendré par ces actions	Modification réelle survenant dans le monde virtuel

Tableau 17. Définition du signe virtuel interactif haut niveau

Désormais, nous pouvons illustrer ce signe à travers plusieurs exemples de jeux vidéo commercialisés pendant la cinquième génération. Dans *Ape Escape* (1999 Sony, PS1), par exemple, le signe interactif *courir* a pour signifiant la valeur d'un *input*, et pour signifié le déplacement du personnage jouable. Par ailleurs, nous pouvons signaler une subtilité dans *Tekken 3* (1998 Namco, PS1) pour le signe interactif *mouvement*, qui réside dans la possibilité d'exécuter plusieurs *inputs* ou combinaisons de touches pour le signifiant. Cela résulte en un signifié qui représente une action de combat contextuelle, pouvant être une attaque ou une esquive. Dans ce même exemple, chaque combinaison complète représente un signe interactif. Nous pouvons également citer *Silent Hill* (1999 Konami, PS1), avec un signe interactif *usage d'une arme* dont le signifiant est représenté par l'exécution d'un *input* et le signifié se manifeste par le fait d'infliger potentiellement des dégâts à une cible. Nous pouvons résumer ces trois signes interactifs dans le tableau ci-dessous, en ignorant le référent car celui-ci n'est pas pertinent pour ce descriptif.

Signe interactif	Signifiant	Signifié
Courir	Valeur d'un *input*	Déplacement du personnage
Mouvement de combat	Exécution d'un ou de plusieurs *inputs*	Action de combat contextuelle (attaque, esquive, etc.)
Usage d'une arme	Exécution d'un *input*	Infliger potentiellement des dégâts à une cible

Tableau 18. Exemples de signes virtuels interactifs haut niveau

Néanmoins, cette combinaison primaire et dyadique entre un signifiant virtuel et un signifié virtuel nous semble insuffisante au regard de la complexité technologique des mécanismes interactifs. De surcroît, elle est trop limitative et restrictive pour rendre compte du processus de création d'un mécanisme

interactif. C'est pourquoi nous pouvons également élaborer le concept du signe (virtuel) interactif bas niveau, en s'inspirant des niveaux de langages de programmation que nous avons mentionnés en introduction. Ainsi, le signe interactif haut niveau que nous venons de présenter contient exclusivement le signifiant et le signifié. Celui-ci permet de réaliser des analyses soit sur des moteurs inaccessibles, où nous devons occulter de nombreux détails, soit sur des jeux vidéo commerciaux, comme illustré par nos différents exemples précédents.

En revanche, le signe interactif bas niveau reprend plutôt la subdivision opérée par Hjelmslev et exposée par Chandler, notamment celle du signe linguistique en « quatre catégories de deux paires : la forme de l'expression et la forme du contenu, ainsi que la substance de l'expression et la substance du contenu » (Chandler 2022, 65 TDR)[347]. Plus encore, on peut mettre en exergue la différence entre le signe linguistique saussurien et hjelmslevien où « le signe saussurien se compose des formes d'expression (le *signifiant*) et de contenu (le *signifié*), pour Hjelmslev, l'expression et le contenu ont à la fois une forme et une substance » (Chandler 2022, 65 TDR)[348]. Cette subdivision supplémentaire permet d'obtenir une description plus approfondie des aspects techniques de la création d'un mécanisme interactif dans un moteur de jeu, tout en intégrant une complexité technologique nécessaire au regard de notre problématique.

Cependant, notre objectif à travers ce signe interactif bas niveau est d'élaborer un modèle pouvant à la fois exprimer l'état d'un mécanisme interactif de jeu en version jouable, mais également en version de développement. Pour cela, nous pouvons nous inspirer des *Build Configuration State* (Epic Games 2023, *Build Configuration Reference*) du moteur *Unreal Engine*. Mais, plutôt que d'employer les quatre modes existants pour notre signe bas niveau, nous allons seulement en retenir deux, qui sont les plus pertinents : *Development* et *Shipping*. Et, si nous traduisons la première configuration par *Développement*, nous allons en revanche altérer le terme de la seconde puisque nous utiliserons le mot *Jouable*.

347. « four categories in two pairs: form of expression and form of content, together with substance of expression and substance of content » (Chandler 2022, 65).

348. « the Saussurean sign consists of the forms of expression (the *signifier*) and content (the signified), for Hjelmslev expression and content have both form and substance » (Chandler 2022, 65).

Et, parce que ces deux aspects sont intrinsèquement liés, nous estimons que la version jouable doit refléter la *forme* alors que la version de développement doit plutôt symboliser la *substance*.

Ainsi, nous avons d'abord le plan virtuel du contenu et de l'expression, représentant respectivement le signifié virtuel et le signifiant virtuel. Le plan virtuel du contenu va traiter des instructions de programmation utilisées pour pouvoir enclencher le mécanisme interactif et le réaliser. En ce qui concerne le plan virtuel de l'expression, celui-ci va mettre en exergue l'aspect visible du mécanisme interactif, aussi bien pour les modèles 3D que pour des particules, voire des objets disponibles dans l'éditeur de niveaux. Notons également que nous omettons des exemples pour illustrer ce signe interactif bas niveau car celui-ci est déjà limpide sur le plan technique. De surcroît, on peut constater que la forme jouable du contenu et de l'expression sont identiques à la forme haut niveau. Toujours au regard des précisions sur cet aspect bas niveau, nous considérons qu'une limitation importante de celui-ci existe : son inapplicabilité aux jeux vidéo commercialisés ainsi qu'à notre corpus vidéoludique. Seul le signe interactif haut niveau peut être appliqué au corpus vidéoludique. Nous pouvons donc synthétiser ces multiples concepts grâce au tableau ci-dessous, directement inspiré de celui rédigé par Chandler (2022, 66). Notons surtout que le tableau original présente les concepts et définitions de Hjelmslev, que nous remplaçons par ceux de notre signe interactif bas niveau.

Plan virtuel	**Substance développementale**	**Forme jouable**
Contenu (Signifié virtuel)	Instructions dans un langage de programmation (*C++* ou *Blueprints*)	Résultat engendré par ces actions
Expression (Signifiant virtuel)	*Modèles 3D (Static Mesh* ou *Skeletal Mesh)*, particules (*Niagara), données*	Action(s) pour déclencher un mécanisme interactif

Tableau 19. Définition du signe virtuel interactif bas niveau

Au-delà de la vision haut niveau et bas niveau du signe interactif, il est également important de discriminer les mécanismes interactifs des éléments qui n'en sont pas. Ainsi, nous pouvons observer l'existence de plusieurs interactions pendant des séquences jouables ou non d'un jeu vidéo, mais que l'avatar ne

pourra jamais réaliser. Nous pouvons illustrer ce phénomène à travers plusieurs jeux vidéo, notamment *Syphon Filter 2* (2000 Bend Studio, PS1) et *Syphon Filter 3* (Bend Studio 2001, PS1). Dans le deuxième opus de cette saga, plus précisément lors de la cinquième mission nommée *McKenzie Airbase Exterior*, on observe un personnage non jouable (désormais PNJ) interagir avec un camion, c'est-à-dire conduire ce véhicule. Or, le personnage jouable, Lian Xing, est incapable de réaliser ce mécanisme interactif. Un autre exemple est visible dans le troisième opus, ici dans la cinématique de la première mission, *Fukushima Hotel*. Cette séquence qui précède la mission montre la conduite d'un véhicule en dehors de l'hôtel. Or, dans la phase de jeu, le personnage ne pourra ni sortir de l'hôtel, ni conduire un quelconque véhicule, alors même que cette action est présentée dans une séquence.

Naturellement, on pourrait d'abord argumenter que l'objectif des développeurs n'était probablement pas d'intégrer ces multiples interactions. Cependant, il est important de souligner que notre analyse ne vise pas à déterminer si un mécanisme interactif potentiel est pertinente dans une séquence de *gameplay*. Notre objectif est simplement de vérifier, voire de constater, la présence de mécanismes interactifs potentiels dans un jeu vidéo mais l'impossibilité d'exécuter ceux-ci pour, dans cet exemple, le personnage jouable. De surcroît, nous ne nous préoccupons aucunement de la contextualisation des signes interactifs dans la narrativité d'un jeu vidéo. Nous occultons intégralement cet aspect pour nous concentrer exclusivement sur les mécanismes interactifs. En d'autres termes, le but majeur de cette analyse est simplement de vérifier s'il existe une limitation technologique, c'est-à-dire si une indisponibilité d'un mécanisme interactif est induite par une contrainte technologique ou relève simplement d'un choix de la part des développeurs.

On pourrait également affirmer, et c'est surtout valable pour la cinquième génération de consoles, voire la sixième, que les séquences cinématiques ne sont pas toujours réalisées avec le moteur du jeu. Cela pourrait potentiellement expliquer l'absence des interactions présentées dans la séquence mentionnée plus haut du jeu *Syphon Filter 3* (Bend Studio 2001, PS1). Sur ce point, Gregory justifie un tel choix pour le développement de certains jeux vidéo, en expliquant que les « *in-game cinematic* (IGC), *non-interactive sequence* (NIS) et *full-motion video* (FMV) (…) communiquent des éléments de l'histoire qui ne s'adaptent pas

bien au *gameplay* interactif » (Gregory 2019, 735 TDR)[349]. De surcroît, il effectue une distinction technologique importante puisque les « IGC et NIS désignent typiquement les séquences non interactives qui sont rendues en temps réel par le moteur de jeu (…) [tandis que] les FMV correspondent aux séquences pré-rendues » (Gregory 2019, 735 TDR)[350]. Néanmoins, nous estimons qu'une telle remarque sur une inadaptabilité d'éléments scénaristiques au *gameplay* n'a aucune pertinence au regard de notre étude. En effet, notre objectif est simplement l'observation d'une absence de certains mécanismes interactifs, sans vérifier si ceux-ci sont pertinents ou pas pour le jeu en question.

Désormais, nous avons terminé nos descriptifs du signe interactif virtuel. Naturellement, il existe d'autres variantes du signe proches du signe saussurien, notamment développées par Klinkenberg, mais nous considérons celles-ci comme moins pertinentes au regard de notre problématique. Par exemple, le signe élaboré par Klinkenberg est, d'après Pagès, un « modèle tétradique (à travers un rectangle ou un carré) (...) proposé par J.-M. Klinkenberg [qui] fait du signe une entité constituée d'un stimulus, d'un signifiant, d'un signifié et d'un référent » (Pagès 2019, 98). Cette définition introduit donc le stimulus, qui selon Klinkenberg est « la face concrète du signe (...) transmissible par le canal (...) d'un de nos cinq sens (...) [et] le support actif du signe (...) [où] le sujet entre en relation avec le signe » (Klinkenberg 2000). Dans notre contexte, nous pourrions incorporer ce stimulus dans notre étude pour évoquer la manière dont l'interaction est enclenchée, par exemple la touche ou le joystick d'une manette. Néanmoins, nous avons déjà souligné, en introduction, que cet aspect est plutôt hors de notre cadre d'étude.

Ainsi, à l'aune de notre définition du signe virtuel interactif, haut niveau et bas niveau, nous pouvons développer un nouveau concept qui va nous permettre de mieux appréhender l'abstraction dans la création des mécanismes interactifs, qui va générer des tensions entre limitations et libertés développementales. Il s'agit

349. « in-game cinematic (IGC), non-interactive sequence (NIS) or full-motion video (FMV) (…) communicate story elements that do not lend themselves well to interactive gameplay » (Gregory 2019, 735).

350. « IGC and NIS typically refer to noninteractive sequences that are rendered in real time by the game engine (…) [while] FMV applies to sequences that have been prerendered » (Gregory 2019, 735).

du niveau d'abstraction interactif que nous allons présenter dans la sous-section suivante.

3-2-A. ABSTRACTION TENSIVE

Au regard de la conception des univers 3D, nous avons constaté un passage de la famille des langages *assembleurs* au *C*, un langage possédant *de facto* un plus haut niveau d'abstraction et une importante portabilité. Pour mettre en exergue ce phénomène dans l'évolution de l'industrie vidéoludique, nous pouvons répertorier, dans le tableau ci-dessous, les langages informatiques utilisés pour créer les jeux sur une seule console majeure par génération, depuis la troisième génération de consoles jusqu'à la cinquième incluse. Naturellement, nos sources sont celles mentionnées depuis l'introduction de ce deuxième chapitre, à savoir Altice (2015), Copetti (2023b) et Copetti (2023a).

Génération	Console	Langage de programmation	
3	NES	Assembleur (6502)	
4	SNES	Assembleur (65C816 et SPC700)	
5	Sony PlayStation	Assembleur (MIPS)	C

Tableau 20. Langages utilisés entre la troisième et la cinquième génération

On constate donc un passage au langage *C*, conférant par la même occasion une indépendance vis-à-vis de la console ainsi que la capacité de réutiliser le même code sur une autre plate-forme. Désormais, le code informatique n'est plus intimement lié au CPU d'une console comme c'était le cas pour un langage *assembleur*. Ainsi, la traduction du code source en langage machine via la compilation n'est *de facto* plus lié à une relation bidirectionnelle monadique comme pour l'*assembleur*, mais plutôt une relation nommée polyadique, pour reprendre le terme d'Hébert (2020, 5). En effet, on constate surtout qu'une instruction en *C* peut correspondre à plusieurs instructions en binaires ou en *assembleur*.

Ainsi, cette utilisation du langage *C* représente la création d'une couche d'abstraction importante dans le développement de jeux vidéo. C'est la raison pour laquelle nous pouvons développer une théorie sémio-informatique qui va mettre en exergue cette abstraction technologique, et que nous nommons l'abstraction tensive. Celle-ci reprend quelques concepts du schéma tensif qui, d'après Hébert, est un « dispositif de la sémiotique post-greimassienne (...)

introduit par Jacques Fontanille et Claude Zilberberg » (Hébert 2005, 111), sans pour autant l'intégrer entièrement dans notre étude. Surtout, ces multiples concepts de la tensivité permettent, pour notre approche du DLT, de pouvoir intégrer dans nos analyses la complexité et l'abstraction technique des multiples outils de développement. Précisons que nous nous appuyons sur l'interprétation du schéma tensif par Hébert, qui indique, dans son article, que « nous entendons (...) montrer le potentiel opératoire de ce dispositif, en nous écartant au besoin de son cadre d'origine » (*ibid.*). Nous ne tenons donc pas compte de la théorie d'origine du schéma tensif. C'est pourquoi nous allons simplement redéfinir les trois concepts inédits suivants :

- ► Intensité d'abstraction ;
- ► Extensité technologique ;
- ► Vecteur de standardisation.

Tout d'abord, nous pouvons définir les deux concepts interprétés par Hébert, « intensité et extensité [qui] constituent, respectivement, le plan du contenu (plan des signifiés) et le plan de l'expression (plan des signifiants) » (Hébert 2005, 112). Il s'agit, plus précisément, d'une « combinaison de deux "valences" (ou dimensions), l'intensité et l'extensité (ou étendue) » (*ibid.*, 115). En utilisant notre signe interactif bas niveau, nous pouvons surtout nous concentrer sur la substance développementale, qui serait ici la plus pertinente pour mettre en exergue les multiples technologies. Ainsi, on peut surtout évoquer la complexité des langages de programmation comme étant liés à l'intensité d'abstraction, et le nombre, voire la complexité des modules d'une API ou d'un moteurs de jeu comme étant des signifiants, associés à l'extensité technologique. Néanmoins, il existe un certain rapport entre ces deux concepts, également indiqué par Hébert, et celui-ci semble être une « corrélation (...) inverse (...) [puisque] l'augmentation de l'une des deux valences s'accompagne de la diminution de l'autre » (*ibid.*, 120). Mais nous réévaluerons cette corrélation, puisque notre objectif est justement de mettre au jour le rapport entre l'abstraction des instructions de programmation, autrement dit l'intensité, et les technologies utilisées, ou encore l'extensité. Cela permet de mettre en exergue des limitations éventuelles des instructions de programmation.

Pour mieux définir cela, nous devons néanmoins illustrer et mieux circonscrire notre concept de l'intensité d'abstraction. Celui-ci analyse essentiellement le type

de langage de programmation utilisé, voire le type d'outil, et son rapport avec la création d'un mécanisme interactif. Et, pour mieux refléter l'abstraction, nous décidons de créer une « segmentation triadique (intensité (...) basse, moyenne et élevée » (*ibid.*, 129) directement inspirée par les courbes esthétiques présentées par Hébert. Cette segmentation va correspondre à trois manières de pouvoir utiliser un langage de programmation et *in fine* exécuter des instructions précises :

- ► Intensité d'abstraction basse : usage d'un langage de programmation standardisé, notamment le *C*, mais sans un moteur préconçu ;
- ► Intensité d'abstraction moyenne : utilisation d'un langage standardisé dans un moteur de jeu, par exemple le *C++* avec le moteur *Unreal Engine 4* ;
- ► Intensité d'abstraction élevée : utilisation d'un langage de script comme *UnrealScript* ou *QuakeC*, spécialement élaboré pour le moteur de jeu.

On note que l'usage de *l'assembleur* n'est pas réellement une intensité basse, mais presque ce que nous pourrions nommer l'intensité d'abstraction zéro. Cependant, il s'agirait d'une inexactitude sur le plan matériel, puisque le langage *assembleur* est traduit en langage machine, ne représentant donc pas le niveau le plus bas. Dans tous les cas, l'intensité d'abstraction est intrinsèquement liée à la nature d'un langage de programmation. C'est pourquoi l'usage du langage *C* augmente l'intensité, puisqu'il s'agit d'un langage permettant une meilleure abstraction et portabilité. Mais, dans l'exemple hypothétique d'optimisation déjà souligné pour le jeu *Metal Gear Solid* (1998 Konami, PS1), on observe une tension émerger, puisque *l'assembleur* était nécessaire pour optimiser les parties critiques du jeu.

Il est également possible d'illustrer le deuxième concept de l'extensité technologique, entremêlé au premier concept, où nous cherchons à évaluer la généricité des outils utilisés au-delà du langage de programmation. L'extensité va ainsi étudier les outils utilisés, notamment les API comme *OpenGL* ou *DirectX*, la création de moteurs de jeu comme *Unreal Engine* ou *Quake Engine* ou encore l'utilisation de moteurs de jeu. Notons que la conception de son propre moteur implique souvent l'utilisation des API, mais nous souhaitions surtout opposer la création d'un moteur avec l'utilisation d'un moteur externe. À l'instar de l'intensité, notre segmentation inédite de l'extensité est également en trois parties :

- ► Extensité technologique basse : utilisation d'un moteur de jeu externalisé.
- ► Extensité technologique moyenne : création d'un moteur de jeu réutilisable ;
- ► Extensité technologique élevée : utilisation des API comme *OpenGL* ou *DirectX* pour concevoir un jeu ;

Néanmoins, ces deux concepts sont insuffisants au regard de nos multiples observations, et il est nécessaire d'avoir un troisième aspect qui vérifie non pas la complexité des outils, mais plutôt la standardisation. C'est pourquoi on peut élaborer un nouveau concept : le vecteur de standardisation. Celui-ci implique de mettre en relief la standardisation de nombreux aspects du développement de jeux vidéo lors de la cinquième génération de consoles. Nous pouvons d'abord illustrer ce phénomène par la démocratisation du langage *C*, standardisé et indépendant des machines, par opposition à la famille des langages *assembleurs*. De surcroît, on observe également la standardisation de plusieurs API, dont les deux principales sur ordinateur, qui sont *OpenGL* et *DirectX*, homogénéisant certains aspects du développement d'un jeu vidéo. Cette standardisation des API est si capitale qu'elle perdure en 2024, dans laquelle la conception d'un moteur de jeu n'a pas réellement, selon Adam Serdar, de « norme importante pour les architectures d'un moteur autre que les APIs que les autres systèmes utilisent » (Serdar 2018, 40 TDR)[351].

Plus encore, cette standardisation va s'opérer de manière plus importante avec la conception de multiples moteurs de jeu, notamment le *DOOM Engine*, *Quake Engine*, *Build Engine* ou encore *Unreal Engine*. Il s'agit d'un aspect particulièrement intéressant dans notre étude puisqu'il permet de constater que les quatre moteurs de jeu majeurs cités sont spécifiquement conçus pour développer des jeux vidéo du genre vidéoludique FPS. Ce genre est décrit par Nicolas Courcier, Mehdi El Kanafi et Raphaël Lucas comme étant un « style de jeu à la vue subjective dans lequel le joueur doit combattre des vagues d'ennemis avec une arme à feu » (Courcier, El Kanafi et Lucas 2019, Introduction TDR)[352].

351. « huge standard for engine systems other than the APIs that other systems are using » (Serdar 2018, 40).
352. « subjective-view game style in which the player must fight waves of enemies with a firearm » (Courcier, El Kanafi et Lucas 2019, Introduction).

Plus qu'un simple type de jeu, il est également présenté par Vaughan Young comme ayant « révolutionné la manière dont les gens jouaient et percevaient les jeux vidéo (...) [qui] est devenu un véritable commerce » (Young 2005, xvii TDR)[353]. Sur le plan technique, Gregory liste de nombreuses caractéristiques d'un moteur vraisemblablement optimisé pour les FPS, notamment un « rendu efficace de vastes mondes virtuels en 3D (...) [ou] un mécanisme de contrôle de caméra/ réglage réactif » (Gregory 2019, 15 TDR)[354].

De surcroît, les exigences techniques d'un FPS semblaient, et semblent encore très élevées puisque, toujours selon Gregory, les « jeux à la première personne sont généralement parmi les plus complexes à créer technologiquement » (*ibid.*, 14 TDR)[355]. Par conséquent, on peut également constater avec Gregory que « beaucoup de grandes innovations technologiques de l'industrie vidéoludique ont émergé des jeux de ce genre » (*ibid.* TDR)[356] et, pour notre cas, les moteurs de jeu sont en grande partie liés au développement des FPS. Cela signifie que la standardisation des outils implique *in fine* la normalisation du moteur de jeu, représentant ainsi l'étape ultime de ce processus de standardisation. La conséquence immédiate de cette démarche est celle d'une standardisation du *gameplay* d'un modèle de jeu précis, qui va créer des limitations technologiques inhérentes liées à un genre vidéoludique précis. Nous pouvons ainsi représenter les différents éléments de standardisation ainsi que les occurrences dans la cinquième génération via le tableau ci-dessous.

353. « revolutionized the way people played and viewed computer games (...) [which] became serious business » (Young 2005, xvii).
354. « efficient rendering of large 3D virtual worlds (...) [or] a responsive camera control/ aiming mechanic » (Gregory 2019, 15).
355. « first-person games are typically some of the most technologically challenging to build » (Gregory 2019, 14).
356. « many of the game industry's big technological innovations arose out of the games in this genre » (Gregory 2019, 14).

Vecteur de standardisation	Exemples
Langage de programmation	C89/ C90...
Matériel	Accélération matérielle 3D...
API	DirectX 1.0 (1995), OpenGL 1.0 (1992)...
Moteur de jeu 3D	Quake Engine (1996), Unreal Engine (1998)...

Tableau 21. Exemples de vecteurs de standardisation

En substance, et à l'aune de notre modèle de l'abstraction tensive, nous avons mis en exergue trois variables importantes mais surtout pertinentes pour calculer l'abstraction technologique dans le développement de jeux vidéo 3D temps réel : intensité d'abstraction, extensité technologique et vecteur de standardisation. Cette mesure de l'abstraction va ainsi conditionner la formation des différents mécanismes interactifs, et implique donc une nature d'un ordre différent. Néanmoins, celle-ci nécessite une entité qui va pouvoir déclencher ces signes interactifs, ce qui fait l'objet de notre prochaine sous-section.

3-2-B. INTERACTANT TRIDIMENSIONNEL

Au regard de notre définition des signes interactifs, nous estimons qu'il s'agit de phénomènes existant dans les mondes virtuels, mais qu'il est possible de déclencher exclusivement grâce à une entité, visible ou invisible. Sur ce point, notre objectif est de mieux appréhender les spécificités de ces mêmes signes interactifs, et les éventuelles contraintes imposées par les entités en question. Surtout, pour mieux définir cette entité, on trouve d'abord la terminologie habituelle, notamment avec la notion de *personnage jouable*, qui implique *de facto* une entité anthropomorphique, humaine ou animale, sémantiquement du moins. En d'autres termes, le personnage jouable peut être un être humain, un animal ou une entité quelconque. Cependant, cette même notion est inflexible pour représenter d'autres cas de figure, notamment le contrôle de véhicules. On peut également signaler une autre terminologie, hypothétiquement plus scientifique, celle d'*avatar*, en particulier parce que, selon Di Filippo, « les chercheurs utilisent souvent la notion d'"avatar" pour parler de l'identité dans les mondes numériques et les jeux vidéo » (Di Filippo 2012). Celle-ci semble surtout représenter « un équivalent du personnage contrôlé par le joueur » (*ibid.*). Néanmoins, notre approche du DLT favorise plutôt un ancrage sémio-informatique, raison pour laquelle nous allons utiliser une partie du vocabulaire du schéma actantiel de Greimas.

Ainsi, de ce schéma, nous pouvons retenir la notion d'*actant*, que nous pouvons définir, avec Georges Forestier, comme étant des « éléments qui ont une fonction dans l'action » (Forestier 2017, ch. 4.2). Ceux-ci sont également très flexibles, puisqu'ils « peuvent être des personnages, des groupes de personnages, des abstractions ou même des personnages absents » (*ibid.*). Néanmoins, plutôt que d'utiliser le terme *actant*, nous le remplaçons par interactant virtuel. Celui-ci représente une entité ou un élément qui va permettre d'entrer en interaction avec l'univers virtuel. Surtout, nous pouvons définir de manière conceptuelle, voire inédite, trois types d'interactants :

- Interactant humanoïde ;
- Interactant véhiculaire ;
- Interactant immatériel.

L'intérêt de cette segmentation est de vérifier si cela implique un biais ou une altération dans la conception des mécanismes interactifs et *in fine* certaines limitations inhérentes. De surcroît, cela permet surtout d'examiner si les interactants peuvent mettre en exergue certaines limitations liées aux genres vidéoludiques. Naturellement, on pourrait remettre en question cette catégorisation tripartite, voire indiquer qu'elle induit elle-même un biais. Cependant, l'objectif est plutôt d'évaluer comment on peut interagir avec l'environnement à travers ces trois aspects, ce qui impliquera de vérifier si notre propre segmentation possède une certaine valeur ou si nous pouvons l'abandonner par la suite. Il ne s'agit donc aucunement de la considérer comme étant absolue, puisque l'industrie vidéoludique et la technologie évoluent continuellement, nécessitant de réadapter certains concepts. Ajoutons également que nous allons ignorer les contraintes liées à l'exportation ou l'importation des modèles 3D, y compris celles du format 3D utilisé, comme le FBX. Nous estimons que cela serait trop complexe pour les traiter raisonnablement.

Ainsi, nous pouvons d'abord définir un interactant humanoïde comme étant le contrôle d'un personnage humain ou anthropomorphique. On pourrait considérer que cette appellation est une inspiration des *Humanoid Avatars* du moteur *Unity*, autrement dit du « *Animation System* de *Unity* (...) [qui sont] des caractéristiques spéciales pour travailler avec des personnages humanoïdes, parce (...) [qu'ils] sont si courants dans les jeux » (Unity Technologies 2023, *Humanoid Avatars* TDR)[357]. Précisons d'emblée qu'il existe dans notre corpus vidéoludique de la cinquième génération des variations, particulièrement pour l'interactant humanoïde. Sur ce point, il est possible d'évoquer certaines caractéristiques pouvant modifier à la fois l'interactant mais aussi le déclenchement des interactions.

Nous allons illustrer ce phénomène à travers deux manières principales pour contrôler un interactant humanoïde dans la cinquième génération : les *Tank Controls* et les *3D Controls*. Par exemple, le jeu *Tomb Raider* (Core Design 1996,

357. « Unity's Animation System (...) [which are] special features for working with humanoid characters, because (...) [they] are so common in games » (Unity Technologies 2023, Humanoid Avatars).

PS1) utilise ce système de déplacement bipède nommé *Tank Controls*, qui selon Matt Perez « fait référence à un mouvement relatif à l'orientation du personnage plutôt qu'à celle de la caméra » (Perez 2015 TDR)[358]. Cela se traduit dans le jeu par des commandes, également expliquées par Perez, où « appuyer vers le haut propulse toujours les personnages vers l'avant, tandis que la gauche et la droite tournent sans accélération, [et] appuyer vers le bas fait marcher le personnage en arrière plutôt que de se retourner » (Perez 2015 TDR)[359]. Un même système également utilisé par d'autres jeux, avec parfois quelques variations, notamment *Syphon Filter* (Eidetic 1999, PS1) ou encore *Fear Effect* (Kronos Digital Entertainment 2000, PS1). Néanmoins, certains jeux permettent de basculer entre les deux types de déplacements, notamment *Silent Hill* (1999 Konami, PS1), sans pour autant modifier les multiples signes interactifs du jeu.

Au regard de l'interactant véhiculaire, nous pouvons constater qu'il s'agit de n'importe quelle entité qui représente un véhicule terrien, aérien, maritime ou spatial. On peut notamment citer *Driver 2: The Wheelman is Back* (Atari 2000, PS1) ou encore *Gran Turismo 2* (Polyphony Digital 1999, PS1). Précisons que le degré de réalisme dans la conduite d'un véhicule n'est pas pris en compte par cette notion d'interactant véhiculaire. Cela signifie *de facto* qu'un kart dans *Crash Team Racing* (Naughty Dog 1999, PS1) est identifiable comme étant un interactant véhiculaire, tout comme une voiture dans la simulation de conduite *Gran Turismo 2* (Polyphony Digital 1999, PS1).

Quant à l'interactant immatériel, il s'agit de l'absence de contrôle d'une entité spécifique, avec le contrôle d'une caméra ou d'un indicateur, par exemple une flèche ou un curseur, pour interagir avec des entités ou le décor. Dans la cinquième génération, on peut illustrer ce phénomène avec de rares exemples, notamment *The Unholy War* (Toys for Bob 1998, PS1). Celui-ci est scindé en deux parties majeures : le contrôle et déplacement d'entités sur un plateau par l'intermédiaire d'un interactant immatériel, mais également des combats entre deux opposants, où on contrôle cette fois-ci un interactant humanoïde ou

358. « refers to movement that is relative to the character's orientation rather than the camera's » (Perez 2015).

359. « pressing up always propels characters forward, while left and right rotate with no acceleration, [and] pressing down causes the character to walk backwards instead of turning around » (Perez 2015).

véhiculaire. On précise que ce dernier jeu permet de passer d'un interactant immatériel à un interactant humanoïde sans transition ni écran de chargement, notamment lors de la séquence sur le plateau virtuel tridimensionnel. Ainsi, il s'agit donc d'une potentielle flexibilité pour permettre d'incarner différents interactants, phénomène qu'on peut constater dans d'autres jeux vidéo de la cinquième génération.

Par exemple, la version *PC* du jeu *Harry Potter and the Sorcerer's Stone* (Foundation 9 Entertainment 2000, PC), développée sur la première version du moteur *Unreal Engine*, permet de contrôler de manière alternative un interactant humanoïde, *Harry Potter*, et un interactant véhiculaire, le balai de course Nimbus 2000 utilisé par *Harry Potter*. Le contrôle de plusieurs interactants simultanément est également observable dans quelques jeux de cinquième génération, notamment *Ape Escape* (1999 Sony, PS1). Ainsi, dès le niveau nommé *Futurama: City Park*, il est possible de pouvoir contrôler, pour certaines phases de jeu, à la fois l'interactant humanoïde *Spike*, personnage principal, mais également le second interactant véhiculaire, une voiture téléguidée.

Ainsi, l'intérêt de cette segmentation est de vérifier s'il existe des jeux possédant une actualisation de ces trois interactants, mais surtout s'il est possible de les combiner, c'est-à-dire de créer des relations de complémentarité. Dans le cas contraire, ils seraient en oppositions catégorielles. Nous pouvons donc répertorier quelques jeux vidéo et mettre en exergue les types d'interactants disponibles dans le tableau ci-dessous.

Jeu	Interactant humanoïde	Interactant véhiculaire	Interactant immatériel	Type de relation
007: Tomorrow Never Dies	Oui	Oui	Non	*Opposition binaire*
Ape Escape	Oui	Oui	Non	*Complémentaire*
Driver 2: The Wheelman is Back	Oui	Oui	Non	*Complémentaire*
Gran Turismo 2	Non	Oui	Non	-
Harry Potter and the Sorcerer's Stone	Oui	Oui	Non	*Opposition binaire*
Metal Gear Solid	Oui	Non	Non	-
Silent Hill	Oui	Non	Non	-
Syphon Filter 2	Oui	Non	Non	-
The Unholy War	Oui	Oui	Oui	*Complémentaire et opposition binaire*

Tableau 22. Interactants disponibles dans le corpus vidéoludique

Néanmoins, nous avons observé des différences ayant un faible impact sur le signifiant et le signifié du signe interactif, notamment celle entre les *Tank Controls* et des *3D Controls*. Phénomène qui permet de mettre en lumière ce que nous nommons des variations interactives résiduelles, autrement dit certaines différences ayant un très faible impact sur le signifiant et le signifié du signe interactif. Elles sont surtout invisibles pour le signe interactif haut niveau, mais véritablement analysables pour le signe interactif bas niveau, notamment par la substance développementale du plan virtuel du contenu. Surtout, ces variations interactives résiduelles peuvent mettre en surbrillance des changements induites par l'usage de différents moteurs de jeu, par exemple un *Quake II Engine* ou *Unreal Engine*. Or, si ce changement de moteur ne change aucunement

l'interprétation du signe interactif haut niveau, alors nous estimons qu'une telle variation n'est pas pertinente à analyser, y compris sur le plan technique. En d'autres termes, s'il n'est pas possible de distinguer des différences interactives entre les moteurs, alors nous pouvons analyser un seul et unique moteur. C'est la raison principale pour laquelle la comparaison anachronique serait tout à fait pertinente pour analyser une seule et unique technologie sur le plan diachronique ; un outil que nous utiliserons par la suite. Néanmoins, notre objectif est plutôt de mettre en corrélation les interactants humanoïdes ou véhiculaires avec l'environnement, ici les mondes ouverts. C'est justement l'objet de notre prochaine sous-section.

3-2-C. PRÉLUDE DES MONDES OUVERTS

La démocratisation de la tridimensionnalité en temps réel lors de la cinquième génération permet d'observer de multiples jeux vidéo incorporant la notion de monde ouvert. Dès l'introduction, nous avions déjà mis en exergue deux jeux antérieurs considérés comme étant les premières occurrences des jeux en monde ouvert : *Elite* (Acornsoft 1984, Multi) et *The Legend of Zelda* (Nintendo 1986, NES). Néanmoins, les jeux qui précèdent cette cinquième génération étaient sur des espaces bidimensionnels, mais ceux-ci impliquaient que, d'après Ken Kutaragi, pour « des jeux comme *Zelda*, (...) effacer un écran signifiait que vous passiez à un écran totalement nouveau » (Kutaragi 2018, 55 TDR)[360]. Kutaragi précise également que ce même concept de passer d'un écran à un autre n'existait pas dans les espaces tridimensionnels (*ibid.*).

C'est pourquoi il était nécessaire de repenser intégralement la conception des espaces, pour les adapter à la technologie de la 3D temps-réel, de surcroît éminemment limitée à cette époque. Naturellement, l'interaction était aussi complexifiée et retravaillée pour l'adapter à ces environnements, pour lesquels on pourrait citer *Super Mario 64* (Nintendo 1996, N64). Ce dernier représente un changement drastique avec les consoles précédentes puisque, d'après Rich Stanton, « les jeux Mario en 2D se déplacent de gauche à droite (...) [alors que] Mario 64 propose des environnements ouverts avec des objectifs placés à différents endroits » (Stanton 2015 TDR)[361]. Les environnements devaient donc être intégralement repensés tout en intégrant les limitations technologiques, ce qui semblait être une réelle performance technique, puisque, comme le rappelle Fred Dutton, « Miyamoto a montré que le déplacement libre était possible, même sur la N64 » (Dutton 2012 TDR)[362].

Nous pouvons également citer un autre jeu N64, grâce à Sleep, « Ocarina Of Time (...) qui a permis des avancées importantes vers (...) le jeu en monde ouvert moderne (...) [avec] le cycle jour/ nuit, une météo dynamique et (...) [une] ville en

360. « games like Zelda, (...) clearing a screen meant you shifted to an entirely new screen » (Kutaragi 2018, 55).
361. « 2D Mario games move from left-to-right (...) [while] Mario 64 has open environments with goals placed at different points » (Stanton 2015).
362. « Miyamoto showed that free roaming was possible, albeit on the N64 » (Dutton 2012).

3D dans laquelle on peut se déplacer librement » (Sleep 2022, 111 TDR)[363]. Pour autant, la notion de monde ouvert s'oppose en réalité à celle d'un « niveau linéaire dirigé » (*ibid.*, 113 TDR)[364]. C'est pourquoi, sur le plan sémantique, nous faisons l'hypothèse que le syntagme monde fermé est extrêmement rare, et qu'on oppose plutôt *monde ouvert* et *niveau linéaire*, compris comme étant une opposition catégorielle.

C'est grâce à cette opposition primaire qu'il est possible de développer une première analyse sur la structuration des mondes virtuels. Notre objectif est surtout de pouvoir remplacer les notions de monde ouvert et de monde linéaire par une structure sémio-informatique à la fois plus précise mais également plus proche du développement de jeux vidéo 3D. Un but que nous allons poursuivre, dans les chapitres cinq et six, en utilisant la notion de continuité et discontinuité, et en s'inspirant du carré sémiotique élaboré par Landowski (2004, 51) à partir des travaux de Greimas.

Néanmoins, la création de ce modèle nécessite plusieurs étapes, la première étant de situer ces potentielles continuités et discontinuités dans les jeux vidéo commerciaux, d'une part, mais également dans les moteurs de jeu, d'autre part. L'objectif étant de vérifier la discordance entre les deux et, pour esquisser les premières constatations sur notre corpus vidéoludique, on observe qu'il existe une nomenclature relativement variée pour décrire et désigner les environnements virtuels dans un espace vidéoludique. En effet, on repère dans les jeux ci-dessous des termes comme *Level*, *Stage*, *World*, *Dimension*, *Arena* ou encore *Hub*. Sur ce point, nous pouvons représenter les différentes désignations dans le tableau ci-dessous, en spécifiant à la fois le jeu en question mais également s'il s'agit d'un monde ouvert ou non.

363. « Ocarina Of Time (...) made important steps towards (...) modern open world game (...) [with] a day/night cycle, changing weather and (...) [a] 3D city in which to roam freely » (Sleep 2022, 111).

364. « linear, directed level » (Sleep 2022, 113).

Jeu	Noms	Monde Ouvert
Ape Escape	*Area, Stage, Dimension*	Non
Body Harvest	*Stage/ Country*	Oui
Crash Bandicoot 2: Cortex Strikes Back	*Level, Warp Room*	Non
Driver 2: The Wheelman is Back	*City*	Oui
Legacy of Kain: Soul Reaver	*World, Warp Gates*	Oui
Silent Hill	*City*	Non
Syphon Filter 2	*Mission/ Level*	Non
Tekken 3	*Stages*	Non

Tableau 23. Nomenclature des environnements du corpus vidéoludique

Ainsi, les différentes appellations dans ce corpus vidéoludique sont peu significatives sur le plan technique, et possèdent des ancrages plutôt thématiques. Sans établir une définition des multiples termes, qui ne concernent pas notre recherche, nous pouvons seulement extraire de ce tableau une tendance : l'usage d'un lexique urbain (*City*, *Country*) lorsqu'un élément du corpus est en monde ouvert ou s'en rapproche, par exemple pour *Silent Hill* (1999 Konami, PS1).

Par ailleurs, nous pouvons également préciser les aspects techniques de certains jeux. Par exemple, le jeu *Legacy of Kain: Soul Reaver* (Crystal Dynamics 1999, Multi), était, d'après Sid Shuman, « conçu comme un jeu d'aventure 3D en monde ouvert, similaire à Zelda » (Shuman 2012 TDR)[365]. Au regard des mondes ouverts, celui-ci répondait aux exigences puisqu'il permettait de pouvoir, selon Linneman, « courir d'un bout du monde à l'autre sans voir un seul écran de chargement » (Linneman 2017 TDR) [366] . Pour Linneman, il s'agit d'une caractéristique qui était possible grâce au « Gex engine (...) [qui] offre un système

365. « conceived as an open-world, Zelda-esque 3D adventure game » (Shuman 2012).
366. « run from one end of the world to another without seeing a single loading screen » (Linneman 2017).

de streaming de données entièrement fonctionnel sur une console qui n'était tout simplement pas conçue pour cette tâche » (Linneman 2017 TDR)[367].

On peut également évoquer *Driver: You Are the Wheelman* (Atari 1999, PS1), souvent considéré, d'après les propos d'Hickey, comme étant « le premier jeu de conduite en monde ouvert » (Hickey 2022, 256 TDR)[368]. Une œuvre vidéoludique dont le développement était difficile pour le contexte technologique de l'époque, car, comme l'explique Dutton, « sur la PS One, il semblait impossible d'avoir des mondes ouverts attrayants, puisque la machine (...) offrait un nombre de polygones relativement faible » (Dutton 2012 TDR)[369]. Les conséquences de cette limitation de la console se manifestaient par l'emploi de nombreux stratagèmes pour la création de l'environnement et exposés par Gregory, notamment avec des techniques comme une « utilisation des cartes bidimensionnelles pour les arbres, les collines et les montagnes » (Gregory 2019, 20 TDR)[370]. On peut également évoquer l'opus suivant, *Driver 2: The Wheelman is Back* (Atari 2000, PS1), qui permettait de sortir du véhicule et offrait donc un *gameplay* plus complexe. Ce dernier est considéré par Racketboy comme étant « très influent sur les titres Grand Theft Auto de l'ère PS2 ainsi que sur les autres jeux "bac à sable" qui ont émergé grâce au succès de GTA » (Racketboy 2018 TDR)[371], autrement dit pour les prochaines générations de consoles.

Naturellement, certains jeux contournaient les limitations techniques en intégrant celles-ci harmonieusement. L'exemple le plus notable est celui de *Silent Hill* (1999 Konami, PS1) puisque, selon Perron, « pour faire un rendu des environnement entièrement en 3D temps réel, la Team Silent a dû gérer les (...) [contraintes] de la PlayStation en limitant le champ de vision (...) [avec] du brouillard et de l'obscurité » (Perron 2012, 27 TDR)[372].

367. « Gex engine (...) [which] deliver a fully functional data streaming system using hardware that simply wasn't designed for the task » (Linneman 2017).
368. « essentially the first open-world driving game » (Hickey 2022, 256).
369. « on the PS One, good-looking, free roaming seemed impossible, the machine (...) had a relatively low polygon count » (Dutton 2012).
370. « employing two-dimensional cards for trees, hills and mountains » (Gregory 2019, 20).
371. « very influential on the PS2-era Grand Theft Auto titles and the other "sandbox" games that were spawned from GTA's success » (Racketboy 2018).
372. « to render its fully real-time 3-D environments, Team Silent dealt with (...) the PlayStation [limitations] by limiting the field of vision (...) [with] fog and darkness » (Perron 2012, 27).

En revanche, et au-delà de ce corpus vidéoludique, en utilisant notre précepte de la comparaison anachronique, nous pouvons réaliser une segmentation bipartite des critères de niveaux dans notre corpus développemental : la *dénomination* dans un moteur et les *technologies* disponibles pour charger un ou plusieurs niveaux. Cela permet de constater que l'unité primaire semble inchangée pour un moteur comme *Unreal Engine 4*. Notons que nous avons également décidé d'ajouter certaines informations relatives à *Unreal Engine 5*, afin d'être le plus exhaustif possible. De surcroît, on constate qu'il existe seulement deux notions distinctes : le niveau d'un jeu ainsi que la manière de *streamer* les *assets*, voire les niveaux. Nous pouvons donc retrouver cela dans le tableau ci-dessous.

Niveau (UnrealEd)	**Technologie (Unreal Engine)**	**Classe Unreal Engine (C++)**
Map, Level, World	*Map • World Partition • World Composition*	*UWorld*, *ULevel*

Tableau 24. Typologie des niveaux dans différents moteurs de jeu

Naturellement, nous n'évoquerons pas les multiples algorithmes qui peuvent être utilisés pour réaliser un rendu d'une scène en 3D, car il s'agit d'une donnée techniquement trop précise au regard de notre module de rationalisation abstraite de notre approche du DLT. Cela étant, il est d'abord possible de se focaliser sur les informations du moteur *Unreal Engine 4* pour constater que trois unités lexicales différentes sont utilisées pour désigner l'espace virtuel : *Map*, *Level* et *World*. En analysant d'abord l'éditeur *UnrealEd*, ainsi que la documentation officielle, nous considérons qu'il n'y a pas de distinction limpide entre ces trois versions. Nous pouvons donc seulement établir une distinction hypothétique.

Ainsi, nous estimons que le terme *Map* correspond au fichier local possédant l'extension *UMAP* et généré directement par *Unreal Engine*, et contenant un espace virtuel quelconque. Cela semble être l'unité de base dans le jeu, et la plus petite unité possible. Concernant *Level*, on pourrait croire qu'il s'agit d'un niveau mais celui-ci va également inclure l'interactivité ou les *Actors*. C'est pourquoi nous avons des *Level Blueprints*. Enfin, *World* pourrait être plutôt utile pour décrire des environnements en monde ouvert, notamment *Open World* pour la

création d'une nouvelle *Map*, mais aussi les technologies *World Composition* ou encore *World Partition*.

Ainsi, si nous examinons notre corpus, nous constatons que les nomenclatures des niveaux dans le corpus vidéoludiques sont plutôt disparates pour les mondes linéaires et relativement unifiées pour les mondes ouverts. Néanmoins, sur le plan technologique, le moteur de jeu *Unreal Engine 4* possède pour unité minimale un fichier contenant une *Map*, et de multiples technologies comme *World Composition*. Mais nous avons besoin d'analyses supplémentaires des technologies en question, que nous effectuerons dans les chapitres ultérieurs, pour pouvoir élaborer un modèle sémio-informatique exhaustif sur les continuités et discontinuités.

3-3. SYNTHÈSE DE LA CINQUIÈME GÉNÉRATION

Ce troisième chapitre a traité du développement de jeux vidéo pendant la cinquième génération de consoles, ayant débuté dès 1993 avec l'apparition de trois consoles de salon majeures au fil du temps : la *PlayStation*, la *Nintendo 64* et la *Sega Saturn*. Nous avons surtout présenté la genèse de la véritable 3D temps réel ainsi que les prémices des moteurs de jeu 3D, un historique qui a permis d'élaborer des théories sémio-informatiques fondamentales pour la suite de notre recherche.

3-1. ASSEMBLAGE DES PREMIERS JEUX

Cette première section majeure est consacrée à l'étude de la distribution de la *PlayStation*, l'analyse des langages *assembleurs* et du langage *C*, de la genèse de la première version du moteur *Unreal Engine* et enfin des outils disponibles pour le développement amateur de jeux sur consoles.

En **introduction** de celle-ci, nous avons d'abord évoqué la quatrième génération, notamment le *PC Engine*, la *Genesis* ainsi que la *Super Nintendo* (SNES*)*, mais nous avons porté notre attention exclusivement sur cette dernière console. Le développement de jeux vidéo sur *SNES* exigeait l'usage de deux langages *assembleurs*, notamment *65C816* et *SPC700*. Néanmoins, le portage d'un jeu d'une machine à une autre était particulièrement complexe et fastidieux, car *l'assembleur* est par définition une famille de langage, n'ayant aucune portabilité. Ainsi, les jeux ne pouvaient conserver que les principes des mécanismes interactifs, mais jamais les implémentations puisque celles-ci étaient entièrement dépendantes de la machine ; il fallait donc réécrire le code entier pour porter un jeu d'une machine à une autre. C'est pourquoi l'idée d'un moteur de jeu en *assembleur*, autrement dit d'éléments réutilisables était, *de facto*, difficilement conciliable avec la technologie de l'époque, et hypothétiquement peu pertinente.

Puis la première sous-section **Développement primo-générationnel** fut principalement dédiée à l'analyse du développement sur la console *PlayStation*, spécialement élaborée pour la 3D temps réel (Lendino 2020 ; Bryan 2005). Nous avons surtout constaté que la cinquième génération a permis l'introduction du développement multi-plateforme, c'est-à-dire à la fois sur une console comme la

PlayStation et sur PC. Une possibilité qui résultait de l'introduction du langage *C* dans le développement de jeux, élaboré en 1972 (Darnell et Margolis 1996) et qualifié de haut niveau par rapport aux langages *assembleurs*, puisqu'il était indépendant des plates-formes (McShaffry et Graham 2013). Cela grâce à un nouveau processus ou mode de traduction du code source d'un langage informatique qui se nomme compilation. On ajoute que le langage *C* avait également pour avantage une plus grande rapidité d'exécution (Sebesta 2016). Néanmoins, plutôt qu'un remplacement complet de *l'assembleur*, certaines sources indiquent l'usage d'un langage *assembleur* conjointement avec le *C* pour l'optimisation de certains scripts. Blum (2005) indique au moins trois moyens pour réaliser cette opération : appeler une fonction *assembleur* en *C*, linker le code *assembleur* plutôt que le *C* ou encore intégrer du code assembleur dans le code *C*. Cette dernière opération se nomme *l'assembleur inline*, et on trouve l'exemple hypothétique de *Metal Gear Solid* (1998 Konami, PS1), où l'objectif était l'optimisation de certaines parties critiques.

La deuxième sous-section, **Prototypage des premiers moteurs***,* a d'abord retracé quelques tentatives de conception de moteurs de jeu dès le début des années 1980, notamment l'*Adventure Game Interpreter* ou encore le *Gold Box Engine* (Greenberg 2021 ; Schules, Peterson et Picard 2018). Une opération toutefois difficile puisque les langages *assembleurs* possèdent généralement une relation d'équivalence avec le code machine. De surcroît, ils étaient surtout liés aux plates-formes, entravant une tentative de modularisation ou standardisation au-delà d'une console ou d'une architecture spécifique. Puis, dès les années 1990, on observe une création et standardisation des API comme *OpenGL* ou *DirectX*, mais également une démocratisation majeure de la 3D temps réel sur PC grâce à la 3Dfx Voodoo 1 en 1996 (Akenine-Möller *et al*. 2008). Carmack et Romero ont également développé la notion de moteur de jeu (Romero 2023 ; Lowood 2014) depuis le *Keen Engine* jusqu'au *Doom Engine*. Les premiers moteurs en 3D temps réel étaient donc disponibles à la fin des années 1990, notamment *Quake Engine*, *Quake II Engine* et *Unreal Engine*. On observe surtout une association entre le genre vidéoludique FPS et le moteur de jeu, puisque ces trois moteurs ont été élaborés pour des FPS, même si *Unreal Engine* a également été utilisé pour des jeux d'action-aventure comme *Harry Potter and the Sorcerer's Stone* (2001 KnowWonder, PC).

Enfin, la troisième sous-section intitulée **Obfuscation du développement amateur** fut consacrée au développement amateur sur consoles, relativement difficile d'accès pour les générations précédentes. Nous avons remarqué une distribution d'un outil de développement amateur de la première console *PlayStation*, la *Net Yaroze*. Relativement onéreuse et restreinte (Bryan 2005 ; Chamberlain 2015 ; Dimitris 2020), le développement amateur de jeux sur cette console nécessitait toutefois le même langage que le développement professionnel, ici le *C*. On mentionne également quelques outils disponibles sur PC, notamment *Virtual Reality Studio* basé sur le moteur *Freescape.*

3-2. SÉMIOTISATION INTERACTIVE DES UNIVERS 3D

La deuxième section majeure a élaboré un cadre théorique fondamental, constituant ainsi les fondations de nos théories sémio-informatiques qui seront *in fine* utilisées pour concevoir notre hyperstructure 3D. Celles-ci découlent des analyses de l'émergence des premiers moteurs de jeu en 3D temps réel, et nous permettent surtout de définir l'unité sémio-informatique fondamentale de l'interactivité.

L'**introduction** de cette deuxième section a permis de mettre en place un système de signes sémiotiques consacrés au développement de jeux vidéo : les signes virtuels interactifs, un outil inspiré des signes linguistiques. Nous avons divisé celui-ci en deux : le signe interactif haut niveau, s'appuyant surtout sur le signe saussurien (Saussure 2005 ; Zufferey et Moeschler 2021), et le signe interactif bas niveau, ce dernier utilisant la subdivision opérée par Hjelmslev (Chandler 2022). Si le signe interactif haut niveau est surtout utilisable pour analyser un jeu en version jouable, le signe interactif bas niveau permet de mettre en exergue les aspects du développement de jeux vidéo, par exemple au niveau des scripts pour le *gameplay*.

Et, dès la première sous-section, **Abstraction tensive**, nous avons introduit le concept d'abstraction tensive, en nous appuyant sur l'interprétation de Hébert (2005) des travaux antérieurs de Fontanille et Zilberberg, plus précisément du schéma tensif. Ainsi, nous avons d'abord intégré les concepts d'intensité et d'extensité (Hébert 2005) pour développer les notions d'intensité d'abstraction et d'extensité technologique. La valence ou dimension d'intensité reflète essentiellement la potentielle complexité d'un langage de programmation utilisé,

comme le *C* ou le *C++*, et son rapport avec la création d'un mécanisme interactif. Elle est segmentée de manière triadique, avec une intensité basse, moyenne et élevée, reflétant le niveau de complexité. Ensuite, nous avons développé la deuxième valence d'extensité, celle-ci évaluant la généricité et la quantité des outils utilisés, par exemple les API comme *DirectX* ou encore les moteurs de jeu, notamment *Unreal Engine*. Pour aller au-delà des aspects de complexité des outils avec l'intensité et l'extensité, nous avons également développé le vecteur de standardisation. Celui-ci a pour objectif de mettre en surbrillance la standardisation des multiples outils de création de jeux, notamment le langage *C*, les API et également les moteurs de jeu.

Puis, à travers la deuxième sous-section, **Interactant tridimensionnel**, nous avons repris la notion d'actant du schéma actantiel (Forestier 2017), une même notion qu'on présentera à nouveau lors du chapitre consacré à l'hyperstructure 3D. De l'actant découle notre notion novatrice de l'interactant virtuel, qui représente une entité ou un élément en interaction avec l'univers virtuel. Celui-ci remplaçant *de facto* la notion d'avatar ou de personnage jouable (Di Filippo 2012). Nous avons ensuite réalisé trois variations inédites de cet interactant : interactant humanoïde, interactant véhiculaire et interactant immatériel. Une subdivision permettant de vérifier si l'interactant va *in fine* mettre en relief des limitations liées aux genres vidéoludiques. L'interactant humanoïde, appellation inspirée par *Unity* (Unity Technologies 2023), représente une entité humanoïde contrôlable. Puis l'interactant véhiculaire symbolise un véhicule quelconque, sans considération pour le degré de réalisme de la conduite. Quant à l'interactant immatériel, celui-ci illustre le contrôle d'une caméra ou d'un indicateur visuel, notamment un curseur ou une flèche, permettant d'interagir avec des entités ou un décor.

Enfin, la troisième sous-section **Prélude des mondes ouverts** était centrée sur l'exploration des premiers mondes ouverts de la cinquième génération. Nous avons constaté que la nomenclature vidéoludique (*Stage, Mission, Level, World, City*...) n'est pas liée à la nomenclature technologique du moteur *Unreal Engine*, qui semble plutôt uniforme (*Map, Level, World, World Composition*). Néanmoins, nous approfondirons les notions de continuités et discontinuités dans les chapitres ultérieurs, afin d'élaborer une théorie exhaustive des mondes ouverts.

4. SIXIÈME GÉNÉRATION : INCRÉMENTATION DES MOTEURS 3D

4-1. TECHNOLOGISATION VIDÉOLUDIQUE

L'avènement de la sixième génération de consoles dès 1998 amorce une ère technologique nouvelle, d'abord caractérisée par une amélioration substantielle de la puissance des machines, notamment de la *Dreamcast*, la *PlayStation 2*, la *GameCube*, ainsi que la *Xbox*. Surtout, certaines consoles ont intégré des fonctionnalités similaires à celles des PC, suggérant un objectif potentiel de remplacement de ces derniers. Ainsi, cette génération débuta par la distribution de la nouvelle console de Sega, la *Dreamcast* en novembre 1998 au Japon (Wardyga 2019, 272). Technologiquement en avance au regard des autres machines, la *Dreamcast* possédait, d'après DeMaria, « un modèle 56K intégré, un clavier et une souris optionnelle (...) [et] afin de prendre en charge les fonctionnalités en ligne » (DeMaria 2019b, 342 TDR)[373]. Et, théoriquement, on peut considérer que cette console, d'après Keith Diefendorff, « affiche 3 à 10 fois plus de polygones 3D que le processeur MIPS de 34 MHz de la PlayStation » (Diefendorff 1999, 1 TDR)[374].

Néanmoins, cette console était hypothétiquement la moins performante de cette génération de consoles, notamment au regard de la *PS2* qui semblait être théoriquement capable d'afficher, toujours selon Diefendorff, « un débit de polygones 10 fois supérieur à celui de la Dreamcast » (*ibid*. TDR)[375]. Précisons que, d'après nos différentes sources, la puissance des consoles était surtout décrite, pour le public, via le nombre de polygones qu'il était possible d'afficher à l'écran. Il était surtout question de mettre en exergue les nombres les plus élevés possibles, selon Kent, où « la Dreamcast de Sega affichait 3 millions de polygones par seconde (...) [et la PlayStation 2] pouvait afficher 60 millions de polygones bruts par seconde » (Kent 2001, 560 TDR)[376]. Néanmoins, si nous utilisons la véridiction technico-discursive de notre DLT, cet écart de puissance supposé possède un indice de véridiction technologique relativement faible.

373. « a built-in 56K modem, optional keyboard and mouse (...) [and] to support the online capabilities » (DeMaria 2019b, 342).
374. « delivers 3 to 10 times as many 3D polygons as PlayStation's 34-MHz MIPS processor » (Diefendorff 1999, 1).
375. « more than 10 times the polygon throughput of Dreamcast » (Diefendorff 1999, 1).
376. « Sega's Dreamcast rendered 3 million polygons per second (...) [and the PlayStation 2] could render 60 million raw polygons per second » (Kent 2001, 560).

Cependant, on constate rapidement avec Williams que « Sega (...) a pris la décision d'arrêter la production de la Dreamcast en 2001 et (...) [est devenu] exclusivement un développeur logiciels » (Williams 2017, 200 TDR)[377], marquant ainsi une fin prématurée de la première console de cette sixième génération. Nous éludons les raisons de fin de production de la *Dreamcast*, car cela n'est pas en lien avec notre problématique, mais nous pouvons souligner, grâce à Wardyga, l'existence de « plus de 600 titres (...) officiellement publiés pour la Dreamcast » (Wardyga 2019, 247 TDR)[378], notamment *Shenmue* (Sega 1999, Dreamcast) ou encore *Sonic Aventure 2* (Sega 2001, Dreamcast).

Par la suite, et moins de deux ans après la distribution de la *Dreamcast*, Sony va organiser, d'après Kent, le « lancement japonais de la PlayStation 2 (...) le 4 mars 2000 » (Kent 2021, ch.1 TDR)[379]. L'objectif hypothétique de Sony, grâce à cette nouvelle console, était de remplacer les ordinateurs, puisque l'entreprise « avait présenté (...) une vision d'un futur foyer rempli de PlayStations et n'ayant aucun PC » (Xbox 2021a, 8:00-8:09 TDR)[380]. Cette perspective technologique future était présente, selon Tom Mainelli, jusque dans son « *design* noir élégant [qui] lui donne moins l'apparence d'une console de jeu pour enfant, et plus celle d'un élément qui pourrait bien s'intégrer dans un centre de divertissement à domicile » (Mainelli 2000 TDR)[381]. Plus encore, cela explique probablement l'apparence d'une tour, symbolisant un ordinateur, mais également la connectique de la PS2, avec deux ports *USB* ainsi qu'un port *Firewire*. En plus de sa rétrocompatibilité matérielle avec la première *PlayStation*, elle disposait également d'un disque dur au Japon, nommé *PlayStation HDD*, et d'une possibilité de se connecter à internet. De surcroît, on constate que la *PlayStation 2* a également « brisé le moule des consoles de jeu en incluant la prise en charge de la lecture des DVD » (CNET Australia 2008 TDR)[382], toujours dans cette

377. « Sega (...) made the decision to discontinue the Dreamcast in 2001 and (...) [became] solely a software developer » (Williams 2017, 200).
378. « more than 600 titles (...) officially published for the Dreamcast » (Wardyga 2019, 247).
379. « Japanese launch of PlayStation 2 (...) on March 4, 2000 » (Kent 2021, ch.1).
380. « was showing (...) a future home that was full of PlayStations and didn't have any PCs in it » (Xbox 2021a, 8:00-8:09).
381. « sleek black design [which] makes it look less like a game machine for kids, and more like something that would fit nicely in a home entertainment center » (Mainelli 2000).
382. « broke the gaming console mold by including support for playing DVDs » (CNET Australia 2008).

perspective de la présenter comme un centre de divertissement plus qu'une simple console de jeux vidéo. Ce phénomène, qui caractérise les consoles telles que la *PlayStation 2*, permet de les décrire comme étant des produits convergents, c'est-à-dire, et pour reprendre la définition de Tripat Gill, des « produits (...) [qui sont] formés en ajoutant une nouvelle fonctionnalité (provenant d'une autre catégorie) à un produit de base existant » (Gill 2008, 46 TDR)[383].

C'est au même moment qu'un acteur majeur dans le monde des ordinateurs et des systèmes d'exploitations intégra le marché vidéoludique, l'entreprise Microsoft, surtout connue pour avoir « proposé le concept de Windows depuis 1983 (Xbox 2021b, 3:17-3:22 TDR)[384]. Et, au début des années 2000, on trouvait de nombreuses versions des systèmes d'exploitation de la famille *Windows*, et Roger Pedersen précise que « Windows XP est considéré comme la version de Windows la plus importante depuis Windows 95 » (Pedersen 2003, 129 TDR)[385]. Un élément d'un intérêt majeur pour le contexte de l'industrie vidéoludique, puisque l'objectif de Microsoft était hypothétiquement de rapprocher le marché des jeux vidéo sur consoles de celui des ordinateurs.

Et le succès important de *Windows* a inspiré la conception d'un système d'exploitation nommé Xbox OS pour la future console de Microsoft puisque, selon DeMaria, « au début du projet [Xbox], (...) [il y avait] quatre options : créer quelque chose à partir de rien (...), utiliser le code source de Win9x (...) utiliser WinCE (...) ou utiliser NT [donc] "nous avons décidé d'utiliser NT" » (DeMaria 2019a, 119 TDR)[386]. En d'autres termes, l'objectif principal, notamment illustré par cette décision interne, était bel et bien de fabriquer une console ayant une forte similitude au fonctionnement d'un ordinateur, aspect hypothétiquement inhabituel dans l'industrie vidéoludique. Une même impression renforcée par le fait que, « à cette époque [NDT : 2001], les consoles de jeu japonaises avaient

383. « products (...) [which are] formed by adding a new functionality (from another category) to an existing base product » (Gill 2008, 46).
384. « came up with the Windows concept back in 1983 (Xbox 2021a, 3:17-3:22).
385. « Windows XP is touted as the most important version of Windows since Windows 95 » (Pedersen 2003, 129).
386. « at the beginning of the [Xbox] project, (...) [there was] four options: create something from scratch (...), use the Win9x source base (...) use WinCE (...) or use NT [so] "we decided to use NT" » (DeMaria 2019a, 119).

des chipsets propriétaires (...) [mais] il y avait une meilleure façon de faire cela, nous allions utiliser l'architecture PC » (Xbox 2021a, 12:00-12:19 TDR)[387].

Cela se manifestait à travers l'intégration de plusieurs composants informatiques initialement exclusifs au PC, et particulièrement un « qui n'avait jamais été fait auparavant sur une console de jeu, (...) un disque dur » (Xbox 2021a, 27:43-27:48 TDR)[388]. Ainsi, nous pouvons constater, grâce à James Batchelor, que la console *Xbox* a été « lancée le 15 novembre 2001 » (Batchelor 2021 TDR)[389]. De surcroît, cette même console proposait nativement des fonctionnalités en ligne, et c'est pour cette raison, selon DeMaria, que le « Xbox Live a été officiellement lancé le 14 novembre 2002 (...) [et] ils ont vendu 150 000 Xbox Live Starter Kits rien que la première semaine » (DeMaria 2019a, 237 TDR)[390].

Par ailleurs, il semblerait que les genres vidéoludiques aient été importants pour Microsoft. En effet, « Ed Fries avait construit Microsoft Game Studios et a organisé la structure du studio par genres [avec] (...) un studio dédié au jeux de rôle, (...) un studio dédié au sport » (Xbox 2021b, 17:27-17:37 TDR)[391]. Sur ce point, nous pouvons citer un jeu considéré comme emblématique pour Barton, « *Halo: Combat Evolved* de Bungie, [qui] est (...) l'un des (...) jeux de tir à la première personne (FPS) les plus influents (...) [qui] a donné aux Américains une (...) raison d'acheter la console Xbox de Microsoft » (Barton 2019, 309 TDR)[392].

Par la suite, c'est la console *Nintendo GameCube* qui va s'implanter dans la sixième génération de consoles, qui selon Wardyga a « débuté au Japon le 14 septembre 2001 (...) [et] a été lancée aux États-Unis le 18 novembre » (Wardyga 2019, 255 TDR)[393]. Néanmoins, et pour reprendre les chiffres de Sony cités par

387. « at that time [TN: 2001], Japanese game consoles had very proprietary chipsets (...) there's a better way to do this, we're going to use PC architecture » (Xbox 2021a, 12:00-12:19).
388. « which had never been done before on a game console, (...) a hard drive » (Xbox 2021a, 27:43-27:48).
389. « launched on November 15, 2001 » (Batchelor 2021).
390. « Xbox Live officially launched November 14, 2002 (...) [and] they sold 150,000 Xbox Live Starter Kits in the first week alone » (DeMaria 2019a, 237).
391. « Ed Fries had built Microsoft Game Studios and kind of organized the studio structure by genre [with] (...) roleplaying studio, (...) sports studio » (Xbox 2021b, 17:27-17:37
392. « Bungie's *Halo: Combat Evolved* [which] is (...) one of the (...) most influential first-person shooter (FPS) (...) [which] gave Americans a (...) reason to purchase Microsoft's Xbox console » (Barton 2019, 309).
393. « debuted in Japan on September 14, 2001 (...) [and] launched in the United States on November 18th » (Wardyga 2019, 255).

Alex Calvin, nous concentrerons surtout notre propos sur la console « PlayStation 2 [qui] s'est finalement écoulée à environ 155 millions d'unités à la date du 31 mars 2012, à partir de laquelle Sony a cessé de publier des mises à jour des ventes de la console » (Calvin 2020, TDR)[394]. Nous pouvons incorporer, dans le tableau ci-dessous, les informations de Wardyga (2019, 265) et de Kent (2021), notamment pour faire un constat du nombre de ventes des consoles de cette même génération.

Consoles	Année de distribution	Ventes mondiales approximatives
Sega Dreamcast	1998	Inférieur à 9 millions
Sony PlayStation 2	2000	Supérieur à 155 millions
Nintendo GameCube	2001	Inférieur à 22 millions
Microsoft Xbox	2001	Supérieur à 24 millions

Tableau 25. Ventes mondiales de la sixième génération

Naturellement, cette avancée technologique a transformé non seulement le processus de développement de jeux vidéo, mais surtout l'industrie vidéoludique entière. De surcroît, nous pouvons également évoquer la maîtrise progressive des technologies au sein d'une génération, car, selon Colin Hughes, une « console est essentiellement une plate-forme fixe (...) [et] il y a un long cycle de vie sur le même matériel » (Hughes 2015, 1:00-1:10 TDR)[395]. C'est pourquoi, pour reprendre l'idée de Hughes, le processus d'apprentissage de développement de jeux sur chaque nouvelle génération est considérable, non seulement pour les programmeurs mais également pour les infographistes (Hughes 2015). Cela a deux conséquences majeures : la création de nombreux prototypes au début d'une génération de consoles, mais également l'absence d'exploitation totale de la puissance technologique par les jeux développés au début de la commercialisation de la console. Par exemple, la version prototypique du jeu

394. « PlayStation 2 [which] ended up shifting around 155 million units by March 31, 2012, at which point Sony stopped providing sales updates for the console » (Calvin 2020).
395. « console is basically a fixed platform (...) [and] there's a long-life cycle with the same hardware » (Hughes 2015, 1:00-1:10)

Final Fantasy X (Square Enix 2001, PS2) présente une possibilité de rotation de la caméra à 360°, ainsi qu'un modèle 3D du personnage Tidus avec des vêtements animés (Unseen64 2006), deux caractéristiques qui n'apparaîtront pas dans la version finale du jeu.

En substance, toutes ces informations permettent de contextualiser l'évolution des jeux vidéo non seulement au regard d'une génération de consoles, mais également à l'intérieur de celle-ci, entre les débuts et la fin, lorsque la technologie est maîtrisée. C'est pourquoi il est important d'étudier les multiples technologies qui seront majoritairement utilisées par l'industrie vidéoludique, à commencer par le langage *C++*, au cœur de notre prochaine sous-section.

4-1-A. Incorporation du C++

L'architecture de la console *PlayStation 2* est surtout connue par rapport à son CPU, nommé *Emotion Engine* en référence aux avancées technologiques rendant possible des animations faciales sur les personnages virtuels. L'article en ligne d'Ellie Gibson met bien en exergue une explication provenant de Phil Harrison, l'ancien vice-président exécutif de *Sony Computer Entertainment Europe*, où « l'idée établie par Kutaragi et son équipe d'ingénieurs était que le CPU serait capable (...) de fournir (...) la puissance mathématique nécessaire pour simuler les émotions » (Gibson 2013, 3 TDR)[396]. Néanmoins, au regard de la complexité du développement de jeux vidéo sur cette nouvelle console, on constate, grâce à Kent, l'apparition dès le départ de multiples « rumeurs (...) [qui] circulaient selon lesquelles, malgré toute sa puissance, la PlayStation 2 était particulièrement difficile à programmer » (Kent 2001, 568 TDR)[397].

Une difficulté qui contraste avec la première *PlayStation* puisque, comme Kent va le souligner en s'appuyant sur le discours de Shinji Mikami, « Sony avait fourni une vaste bibliothèque avec la PlayStation (...) [qui] ferait une grande partie du travail, mais avec la PlayStation 2, il n'y a pas de bibliothèque, on doit créer notre propre bibliothèque » (*ibid*. TDR)[398]. Surtout, ces difficultés mises en relief ne semblaient pas être des cas isolés, puisque « les concepteurs chez Namco, Konami et SquareSoft ont tous mentionné les difficultés à programmer sur la PS2 » (Kent 2000 TDR)[399]. Des propos qu'on retrouve également dans des documents relatant le développement du jeu *Jak and Daxter* (Naughty Dog 2001, PS2). En effet, le programmeur Andy Gavin a indiqué que « la PS2 était difficile à programmer (…) [et] en plus de cela, (...) l'intégralité du jeu [a été écrit] dans un langage de programmation conçu par moi-même [NDT : Andy Gavin] et nommé

396. « the idea Kutaragi and his team of engineers established was that the CPU would be capable (...) of delivering (...) mathematical capability required to simulate emotion » (Gibson 2013, 3).
397. « rumors (...) [were] circulating that for all of its power, PlayStation 2 was distinctly hard to program » (Kent 2001, 568).
398. « Sony provided an extensive library with PlayStation (...) [which] would do a lot of the work, but with PlayStation 2, there is no library, we need to create our own library » (Kent 2001, 568).
399 . « designers at Namco, Konami, and SquareSoft all mentioned the difficulties of programming PS2 » (Kent 2000).

GOAL » (Dutton 2012 TDR)[400], spécifiquement élaboré pour ce jeu. Cependant, la conception d'un langage de programmation spécifique a impliqué des difficultés substantielles pendant le développement. En effet, selon Stephen White, l'équipe de développement « a travaillé dans un tel isolement [qu'elle] (..) a abandonné les outils de développement tiers, notamment les profileurs et débogueurs, ainsi (…) que les bibliothèques existantes, y compris le code précédemment développé en interne » (White 2002 TDR) [401] . C'est pourquoi de nombreux développeurs préféraient l'usage de technologies standardisées.

Par ailleurs, nous pouvons mettre en exergue l'introduction du langage *C++*, subrepticement pendant la cinquième génération puis de manière hypothétiquement plus importante lors de cette sixième. Nous rappelons ici que le développement de jeux est relativement secret (Whitson 2018 ; DeLoura 1999), comme exposé lors du deuxième chapitre consacré au DLT. Ainsi, nous pouvons seulement émettre des hypothèses dans certains cas, notamment celui-ci portant sur l'utilisation du *C++*. Ce même langage, comme le rappelle Petzold, a été « créé par l'informaticien danois Bjarne Stroustrup (né en 1950) en 1985 » (Petzold 2023, 433 TDR)[402]. La popularité du *C++* augmentera substantiellement, au point de devenir l'un des langages les plus populaires à l'heure actuelle, raison principale pour laquelle c'est le « C++ [qui] est probablement le langage le plus utilisé dans l'industrie des jeux vidéo » (Gregory 2019, 105 : TDR)[403]. Surtout, on a observé une adoption graduelle du langage, puisque « à mesure que la vitesse des processeurs et des compilateurs a continué à s'améliorer, le C a finalement été remplacé par le C++ dans la plupart des studios de développement de jeux » (McShaffry et Graham 2013, 333 TDR)[404]. Nous pouvons également émettre l'hypothèse que son adoption massive est liée à sa rétrocompatibilité avec le

400. « the PS2 was difficult to program (…) [and] on top of that, (...) the entire game [was written] in a programming language of my own design called GOAL » (Dutton 2012).
401. « worked in such isolation [that it] (...) gave up third-party development tools such as profilers and debuggers, and (…) existing libraries, including code previously developed internally » (White 2002).
402. « created by Danish computer scientist Bjarne Stroustrup¤(born 1950) in 1985 » (Petzold 2023, 433).
403. « C++ [which] is arguably the most commonly used language in the game industry » (Gregory 2019, 105).
404. « as processor speeds and compilers continued to improve, C was eventually replaced by C++ in most game studios » (McShaffry et Graham 2013, 333).

langage *C*, diminuant *de facto* la courbe d'apprentissage. Néanmoins, nous ne possédons pas des sources d'informations scientifiques pouvant attester d'une utilisation massive du langage *C++* pour la plupart des jeux développés précisément sur la PS2. Nous pouvons affirmer qu'il s'imposera après la famille des langages *assembleurs* ainsi que le *C* de la cinquième génération, mais la période d'adoption générale est incertaine.

De surcroît, Horton et Van Weert estiment qu'il « est l'un des plus vieux langages de programmation encore très utilisé » (Horton et Van Weert 2020, 1 TDR)[405]. Au regard de cette popularité supposée, nous pouvons nous appuyer sur différents classements, notamment l'index TIOBE de décembre 2023 (TIOBE 2023), le classement RedMonk de janvier 2023 (O'Grady 2023) ou encore IEEE Spectrum (Cass 2023) pour le tableau ci-dessous.

Rang	**IEEE Spectrum** (2023)	**Index TIOBE** (Décembre 2023)	**RedMonk** (Janvier 2023)
1	Python	Python	JavaScript
2	Java	C	Python
3	**C++**	**C++**	Java
4	C	Java	PHP
5	JavaScript	C#	C#

Tableau 26. Trois classements de popularité des langages de programmation

Cela étant, le *C++* a surtout été élaboré, d'après Liang *et al.*, comme représentant « une extension du [langage] C » (Liang *et al.* 2014, 33 TDR)[406]. Et l'objectif principal était, selon Stroustrup, « d'apporter les facilités [du langage] Simula pour l'organisation d'un programme avec la flexibilité et l'efficacité du C pour la programmation système » (Stroustrup 2022 TDR)[407]. Cet aspect d'*organisation* d'un programme évoqué par Stroustrup inclut en réalité le concept des paradigmes de programmation, une théorie capitale constituant l'une des raisons de l'élaboration du *C++*. De surcroît, pour recontextualiser l'idée d'un

405 « is one of the oldest programming languages still in active use » (Horton et Van Weert 2020, 1).

406. « an extension of [the] C [language] » (Liang *et al.* 2014, 33).

407. « to provide Simula's facilities for program organization together with C's efficiency and flexibility for systems programming » (Stroustrup 2022).

paradigme, on précise avec Sebesta que, « à la fin des années 1970, un passage des (...) programmes orientés procédures à ceux orientés données a commencé (...) [et la prochaine] évolution (...) au début des années 1980 était la conception orientée objet » (Sebesta 2016, 43 TDR)[408]. Néanmoins, Clark précise que l'émergence du paradigme orienté objet est presque aussi ancienne que la naissance des premiers jeux vidéo, puisque les « concepts de la POO ont commencé à émerger au milieu des années 1960 avec un langage de programmation appelé Simula et ont ensuite évolué dans les années 1970 avec l'arrivée du Smalltalk » (Clark 2013, 2 TDR)[409]. Par ailleurs, les programmes orientés procédures sont élaborés avec des langages comme le *C*, et Grégoire précise que « l'unité de programmation principale dans le paradigme procédural est la procédure ou la fonction » (Gregoire 2021, 422 TDR)[410].

Désormais, nous pouvons mettre en exergue ces multiples paradigmes, et on peut constater avec Sebesta que les « langages de programmation sont souvent classés en quatre catégories : impératifs, fonctionnels, logiques et orientés objet » (Sebesta 2016, 44 TDR)[411] ; ces catégories étant les paradigmes en question. D'autres classifications existent, mais elles ne concernent pas notre thèse et n'ont aucun impact sur notre problématique. De surcroît, afin d'analyser quelques caractéristiques du langage *C++*, nous pouvons principalement nous concentrer sur le paradigme orienté objet. Précisons néanmoins que plusieurs paradigmes sont utilisables pour le *C++* en plus de l'orienté objet, notamment l'impératif et le procédural, ce qui signifie que, selon Dawson, « C++ est un langage de programmation multi-paradigme » (Dawson 2014, 297 TDR)[412]. Et, pour présenter succinctement une définition plus formelle d'un paradigme, il est possible d'affirmer avec Gregoire que « la programmation procédurale ou la programmation orientée objet (...) représentent simplement différentes façons

408. « in the late 1970s, a shift from procedure-oriented to data-oriented program (...) began (...) [and the next] evolution (...) in the early 1980s, is object-oriented design » (Sebesta 2016, 43).
409. « OOP concepts started surfacing in the mid-1960s with a programming language called Simula and further evolved in the 1970s with advent of Smalltalk » (Clark 2013, 2).
410. « the main programming unit in the procedural paradigm is the procedure or function » (Gregoire 2021, 422).
411. « programming languages are often categorized into four bins: imperative, functional, logic, and object oriented" (Sebesta 2016, 44).
412. « C++ is a multi-paradigm programming language » (Dawson 2014, 297).

de raisonner sur (...) [un] programme » (Gregoire 2021, 169 TDR)[413]. Mais nous reviendrons sur cet aspect dans le chapitre suivant.

Ainsi, nous pouvons considérer, grâce aux propos de Clark, que la « programmation orienté objet est une approche du développement logiciel dans laquelle la structure du logiciel repose sur des objets interagissant les uns avec les autres pour accomplir une tâche » (Clark 2013, 1 TDR)[414]. Ces multiples objets qui interagissent présentent l'avantage de pouvoir s'adapter à une meilleure modélisation de la réalité. En effet, et selon Horton et Van Weert, cela contraste avec la « programmation procédurale, qui implique (...) une solution en termes de types de données fondamentaux (...) [et] ne permet pas de modéliser de manière très efficace des objets du monde réel » (Horton et Van Weert 2020, 395 TDR)[415]. Et ces objets sont, pour le paradigme orienté objet, des entités qui représentent des éléments du monde réel ou virtuel. Dawson va préciser la notion d'objet dans les univers vidéoludiques puisque, « que (...) [nous sommes] en train d'évoquer des vaisseaux spatiaux extra-terrestres, des flèches empoisonnées ou des poulets mutants colériques, les jeux regorgent d'objets » (Dawson 2014, 270 TDR)[416].

On précise toutefois que les objets sont créés à partir de classes. Et, pour définir cette dernière notion, on peut affirmer qu'une « classe est comme un plan, [et] tout comme un plan définit la structure d'un bâtiment, une classe définit la structure d'un objet » (*ibid.*, 272 TDR)[417]. Il est également possible de définir une classe comme étant un « *contrat* qui définit les champs de données et les fonctions d'un objet, [et] un objet est une instance d'une classe » (Liang *et al.* 2014, 362 TDR)[418]. Et, au regard des caractéristiques de ce paradigme, les

413. « procedural programming or object-oriented programming (...) just represent different ways of reasoning about (...) [a] program » (Gregoire 2021, 169).
414. « object-oriented programming is an approach to software development in which the structure of the software is based on objects interacting with each other to accomplish a task » (Clark 2013, 1).
415. « procedural programming, which involves (...) a solution in terms of fundamental data types (...) [and] don't allow you to model real-world objects (...) very well » (Horton et Van Weert 2020, 395).
416. « whether (...) [we are] talking about alien spacecrafts, poisonous arrows, or angry mutant chickens, games are full of objects » (Dawson 2014, 270).
417. « class is like a blueprint, [and] just as a blueprint defines the structure of a building, a class defines the structure of an object » (Dawson 2014, 272).
418. « *contract* that defines what an object's data fields and functions will be, [and] an object is an instance of a class » (Liang *et al.* 2014, 362).

définitions convergent habituellement, selon Horton et Van Weert, vers quatre « idées fondamentales (notamment l'encapsulation, l'abstraction, l'héritage et le polymorphisme) » (Horton et Van Weert 2020, 396 TDR) [419]. Ces notions possèdent de nombreuses subtilités et variantes, mais nous estimons qu'elles sont en dehors du cadre imposé par notre problématique.

Dans tous les cas, les implications du paradigme orienté objet sont nombreuses, et nous n'avons qu'effleuré sa complexité. De surcroît, nous éludons les multiples variantes du paradigme orienté objet, par exemple celle où, selon Robert Nystrom, « en ce qui concerne les objets, il en existe en réalité deux approches, les classes et les prototypes » (Nystrom 2021, 3.9.3 TDR)[420], puisque cela ne concerne pas réellement le *C++*. Enfin, nous pouvons également préciser que le concept de classe peut être rapproché d'une notion linguistique, puisque Chandler indique que « la grande majorité des mots lexicaux (...) existent à un niveau élevé d'abstraction et font référence à des classes de choses (telles que des *bâtiments*) ou de concepts (comme la *construction*) » (Chandler 2022, 71 TDR)[421]. Désormais, plus que le langage de programmation en lui-même, c'était surtout un autre outil majeur qui a connu une démocratisation massive dans la sixième génération de consoles : les moteurs de jeu.

419. « important ideas (famously encapsulation, data hiding, inheritance, and polymorphism) » (Horton et Van Weert 2020, 396).

420. « when it comes to objects, there are actually two approaches to them, classes and prototypes » (Nystrom 2021, 3.9.3).

421. « the vast majority of lexical words (...) exist on a high level of abstraction and refer to classes of things (such as *buildings*) or to concepts (such as *construction*) » (Chandler 2022, 71).

4-1-B. DÉVELOPPEMENT DES MOTEURS DE JEU AAA

Les exigences de conception des œuvres vidéoludiques étant plus élevées lors de la sixième génération, le développement s'accompagnait à la fois d'une meilleure maîtrise des technologies mais aussi de la volonté de mieux répondre aux attentes des joueurs. Ainsi, les développeurs étaient désormais partagés entre deux possibilités majeures : la conception d'un moteur de jeu *from scratch* et l'utilisation d'un moteur existant. Cela contraste avec la cinquième génération, puisque les moteurs semblaient être relativement rares, où *Unreal Engine* et *Quake Engine* constituaient plus une exception qu'une règle pour la création de jeux vidéo en 3D temps réel.

Naturellement, le choix exposé entre la création d'un moteur et l'utilisation d'un moteur existant est cornélien, puisque le développement d'un moteur est un projet si important, nous indique Pedersen, que « concevoir et programmer un moteur de jeu 3D de pointe pourrait prendre un an ou plus à développer pour une équipe de programmeurs » (Pedersen 2003, 165 TDR)[422]. Si nous pouvons nuancer un éventuel laps de temps pour le développement d'un moteur de jeu, qui n'est pas nécessairement quantifiable sans illustrer son propos par des exemples concrets, la conception d'un moteur nous semble être un travail véritablement chronophage. Surtout, pour des développeurs, Tommy Refenes indique que le fait de « réaliser un moteur (...) [est] un grand défi ; même quelque chose de simple comme l'affichage d'un triangle à l'écran prend 100 lignes de code OpenGL ou 200 lignes de code DirectX » (Refenes 2018, 89 TDR)[423].

En revanche, parallèlement à la démocratisation importante du langage de programmation *C++*, on peut observer une commercialisation d'un grand nombre de moteurs de jeu destinés aux développeurs professionnels dès la sixième génération. C'est précisément ce phénomène qui est mis en exergue par l'ouvrage de Pedersen, qui va définir trois catégories de moteurs de jeu disponibles pendant cette période : économiques, moyens et luxueux. (Pedersen

422. « Designing and programming a state-of-the-art 3D game engine could take a team of programmers a year or more to develop » (Pedersen 2003, 165).

423. « making an engine (...) [is] a huge challenge; even getting something simple like a triangle drawing on the screen takes 100 lines of OpenGL code or 200 lines of DirectX code » (Refenes 2018, 89).

2003, 165). Une classification tripartite qui semble axée sur la supposée qualité desdits moteurs, où « moins le moteur est cher, plus la technologie est ancienne et moins il y a de caractéristiques disponibles » (*ibid*. TDR)[424]. De surcroît, l'usage de moteurs de jeu ne semble pas réduire le temps de développement. En effet, on peut surtout constater, grâce à Olsen, Kirmse et Sánchez-Crespo, que la « technologie sous licence n'a pas permis d'économiser plusieurs mois de temps de développement (...) ce temps et ces efforts ont finalement été redistribués dans d'autres aspects » (Olsen, Kirmse et Sánchez-Crespo 2001, 32 TDR)[425], permettant hypothétiquement la création de plus de contenu pour un même jeu.

Par ailleurs, la définition explicite d'un moteur de jeu pour Pedersen paraît incertaine. En effet, la liste en question semble mélanger plusieurs types de moteurs que nous avons cités en amont, autrement dit des moteurs de jeu avec des moteurs de rendu 3D, voire des outils similaires à des API. Or, il s'agit de technologies différentes, comme cela a été observé dans le chapitre précédent portant sur la cinquième génération. Néanmoins, nous pouvons présenter une version synthétisée du classement proposé par Pedersen (2003) dans le tableau ci-dessous.

Type de moteur	Exemple
Moteur économique	Genesis 3D, Power Render 4 Engine, Tribes 2/ Torque Engine, Quake Engine, Quake 2 Engine.
Moteur moyen	LithTech Engine, Nocturne Engine, Serious Engine, Unreal Engine 2.
Moteur luxueux	NetImmerse 3D Game Engine, Renderware Platform, RenderWare Studio, Quake 3 Arena Engine.

Tableau 27. Classement des moteurs selon Pedersen (2003)

Au-delà d'une classification relativement subjective, nous pouvons cependant observer qu'il existe un grand nombre de moteurs de jeu disponibles, contrairement à la génération précédente. C'est pourquoi nous pouvons, d'abord, mieux circonscrire la définition d'un moteur de jeu, en nous appuyant notamment

424 « the less expensive the engine, the older the technology and less features there are available » (Pedersen 2003, 165).

425. « licensed technology didn't save the months of (...) development time, (...) that time and effort ended up redistributed to other areas » (Olsen, Kirmse et Sánchez-Crespo 2001, 32).

sur celle de Gregory. Ainsi, un moteur de jeu contient potentiellement, d'après Gregory, les éléments suivants : « un moteur de rendu, un moteur physique et de collision, un système d'animation, un système audio, un modèle de *Game Object*, un système d'intelligence artificielle, etc. » (Gregory 2019, 3 TDR)[426]. On peut ajouter, grâce à Politowski *et al.*, que ce même moteur peut être « conçu en partant de rien pendant le développement d'un jeu, réutilisé d'un jeu précédent, modifié d'un moteur en Open Source, ou acheté » (Politowski *et al.* 2020, 1 TDR)[427]. Néanmoins, il était d'usage, et c'est encore le cas pour certains studios dans les années 2020, de créer des outils *ex nihilo* d'après Gregory, puisque « de nombreuses entreprises développent et maintiennent des moteurs de jeu internes et propriétaires » (Gregory 2019, 35 TDR)[428].

De surcroît, et particulièrement pour les moteurs internes et propriétaires, on peut ajouter avec Gregory que la « frontière entre un jeu et son moteur est souvent floue, [et] certains moteurs établissent une distinction relativement claire, tandis que d'autres ne font presque aucun effort pour les séparer » (Gregory 2019, 12 TDR)[429]. Par conséquent, chaque moteur est *de facto* développé d'une manière unique, et cela implique pour McShaffry et Graham qu'il existe presque « autant de chemins pour assembler les sous-systèmes d'un jeu qu'il y a de programmeurs de jeux vidéo » (McShaffry et Graham 2013, 26 TDR)[430].

Cependant, s'il existe de nombreux moteurs, voire de manières d'élaborer ceux-ci, nous pouvons désormais contraster avec la nomenclature de Pedersen en s'appuyant sur les propos d'Aras Pranckevičius, indiquant que, « vers 2002 ou 2003, les seuls moteurs qui étaient accompagnés de véritables outils étaient

426. « the rendering engine, the collision and physics engine, the animation system, the audio system, the game world object model, the artificial intelligence system and so on » (Gregory 2019, 3).
427. « built from scratch during game development, reused from previous games, extended from open-sources ones, or bought off the shelves » (Politowski *et al.* 2020, 1)
428. « many companies build and maintain proprietary in-house game engines. » (Gregory 2019, 35).
429. « line between a game and its engine is often blurry, [and] some engines make a reasonably clear distinction, while others make almost no attempt to separate the two. » (Gregory 2019, 12).
430. « as many ways to assemble the subsystems of a game as there are game programmers » (McShaffry et Graham 2013, 26).

Render-Ware et Unreal Engine 2 » (Pranckevičius 2018, 45 TDR)[431]. Ces deux moteurs semblaient être les plus utilisés dans cette génération, puisque Stu Horvath signale que le moteur « Unreal 2 (...) a finalement été utilisé pour créer plus de 100 titres » (Horvath 2012 TDR)[432] de jeux vidéo. Un usage éminemment important dans l'industrie vidéoludique, ce qui était également le cas des outils estampillés *RenderWare*, qui étaient également, d'après Pedersen, « utilisé de manière importante par la majorité des développeurs et éditeurs à l'intérieur de la communauté de développement de jeux vidéo (...) dans plus de 250 titres » (Pedersen 2003, 180 TDR)[433].

Nous pouvons donc succinctement présenter les outils *RenderWare*, relativement anciens d'après Boutellier, Gassmann et von Zedtwitz puisque « l'entreprise appelée Criterion Software Ltd (CSL) a été créée en 1993 pour l'exploitation commerciale de RenderWare » (Boutellier, Gassmann et von Zedtwitz 2008, 331 TDR)[434]. Et ces outils ont été « distribués (...) à plus de 1 000 utilisateurs entre 1994 et 1997 (...) [et] à la fin de l'année 1995, CSL a diversifié son activité de rendu graphique 3D dans le domaine des jeux 3D » (*ibid.*, 333 TDR)[435]. Nous l'avons toutefois omis dans notre chapitre portant sur la cinquième génération de consoles, car son utilisation s'est principalement démocratisée pendant cette sixième génération.

Néanmoins, Pedersen précise qu'il existe « deux produits RenderWare, RenderWare Platform et RenderWare Studio » (Pedersen 2003, 180 TDR)[436]. Ceux-ci sont d'une nature bien distincte : *RenderWare Platform* semble être une API complète, un « ensemble de technologies de production de jeux véritablement multiplateforme » (*ibid.* TDR)[437]. On peut également décrire ceux-

431. « around 2002 or 2003, the only engines to come with actual tools were Render-Ware and Unreal Engine 2. » (Pranckevičius 2018, 45).
432. « Unreal 2 (...) would eventually be used to create more than 100 titles » (Horvath 2012).
433. « used extensively by the majority of developers and publishers within the game development community (...) in more than 250 titles » (Pedersen 2003, 180).
434. « company called Criterion Software Ltd (CSL) was set up in 1993 for business exploitation of RenderWare » (Boutellier, Gassmann et von Zedtwitz 2008, 331).
435. « distributed (...) to more than 1,000 users from 1994 to 1997 (...) [and] in late 1995, CSL diversified its 3D graphics business into the area of 3D games » (Boutellier, Gassmann et von Zedtwitz 2008, 333).
436. « two RenderWare products, RenderWare Platform and RenderWare Studio » (Pedersen 2003, 180).
437. « suite of truly multiplatform game production technologies » (Pedersen 2003, 180).

ci comme un ensemble de *middlewares*, une désignation, selon Heather Maxwell Chandler et Rafael Chandler, qui représente un « terme générique désignant un ensemble logiciel prêt à l'emploi qui peut être modifié pour une utilisation dans un jeu » (Chandler et Chandler 2011, 198 TDR)[438]. On peut également illustrer cette définition à travers de multiples exemples de Chandler et Chandler, qui peuvent rappeler les différents composants énumérés par Gregory, notamment des « systèmes de jeu, notamment l'AI, l'animation, la physique, le rendu et le réseau » (*ibid*. TDR)[439]. S'il est précisé que les « moteurs de jeu multi-plate-forme (...) sont considérés comme étant des *middlewares* » (*ibid*., 199 TDR)[440], nous nous permettons de nuancer quelque peu le propos.

En effet, nous considérons qu'un *middleware* est simplement un système particulier et indépendant à l'intérieur d'un moteur, par exemple un système de rendu ou un moteur physique, et non pas un système intégralement unifié. C'est ainsi que *RenderWare Platform* nécessite d'assembler les différents *middlewares* indépendants, et il n'existe pas de système unifié déjà prêt à l'emploi à l'intérieur de cette version. En revanche, on peut contraster cette version avec la seconde, c'est-à-dire « RenderWare Studio [qui] est le seul système de production de jeu » (Pedersen 2003, 180 TDR)[441]. Cette définition semble être plus proche d'un moteur de jeu que d'un *middleware*, selon la description de Pedersen. Au regard des éléments exposés, on peut également constater avec Chris Chapman une grande flexibilité des outils *Renderware*, qui semblaient être particulièrement « polyvalent (...) [pour] concevoir un jeu de sport, un jeu de tir, un jeu de stratégie, un casse-tête, un jeu de plate-forme, un RPG, une simulation de vol, [et] il y avait même un ou deux MMO » (Chapman 2020, 2:56-3:06 TDR)[442].

438. « all-encompassing term for an out-of-the-box software package that can be modified for use in a game » (Chandler et Chandler 2011, 198).
439. « game systems, such as AI, animation, physics, rendering, and networking » (Chandler et Chandler 2011, 198).
440. « cross-platform game engines (...) are considered middleware » (Chandler et Chandler 2011, 199).
441. « RenderWare Studio [which] is the only complete game production system » (Pedersen 2003, 180).
442. « versatile (...) [to] build a sports game, a shooter, a strategy game, puzzler, platformer, RPG, flight sim, [and] there were even one or two MMOs » (Chapman 2020, 2:56-3:06).

Quant au moteur *Unreal Engine 2*, et selon *Epic Games*, celui-ci était hypothétiquement un véritable moteur de jeu unifié, à l'instar de *RenderWare Studio*, c'est-à-dire un « *Framework* de développement de jeu complet » (Epic Games 2005 TDR)[443]. De surcroît, il paraissait être particulièrement flexible, selon les propos de Thomsen, dans la mesure où, « avec l'avènement d'Unreal 2.0, dire que quelque chose était créé sur un moteur particulier ne dictait pas nécessairement le type d'expérience (...) [qu'un joueur] aurait eu » (Thomsen 2012 TDR)[444].

Par exemple, nous pouvons évoquer les Éditeurs de niveaux, déjà mentionnés dans la génération précédente, comme *UnrealEd* ou *DoomEd*. On peut décrire cela comme étant des WYSIWYG, c'est-à-dire « un environnement (*What You See Is What You Get*) pour rassembler les *assets* et traiter rapidement les problèmes de design ou du moteur » (Arnaud 2011, 25 TDR)[445]. De surcroît, la page des caractéristiques du moteur *Unreal Engine 2* indique bien que « Unreal Editor (UnrealEd) est un pur outil de création de contenu "*What You See is What You Get*" » (Epic Games 2005 TDR)[446]. Naturellement, la présence ou l'absence d'un Éditeur de niveaux n'entre nullement en compte dans notre définition d'un moteur de jeu. Ainsi, certains moteurs peuvent posséder un éditeur de niveaux, tandis que d'autres n'en ont pas, tout en étant bel et bien des moteurs de jeu.

Précisons également que le moteur *Unreal Engine 2* permettait ce qu'il est possible de nommer *modding*, un sujet que nous omettons dans notre recherche, puisque cela n'est pas lié à notre problématique. D'abord, nous pouvons définir cela, grâce à Andy Dyer, comme étant « toute modification apportée au jeu de base d'origine, que ce soit une légère modification ou alors une réinvention complète » (Dyer 2016 TDR)[447]. De surcroît, comme mis en relief précédemment, et également indiqué par Bilińska-Reformat, Dewalska-Opitek et Hofman-Kohlmeyer, « les clients s'engagent dans le *modding* de jeux, mais le niveau

443. « complete game development framework » (Epic Games 2005).
444. « with the advent of Unreal 2.0, saying something built on a particular engine didn't necessarily dictate the kind of experience (...) [a gamer] would have » (Thomsen 2012).
445. « (What You See Is What You Get) environment for putting together the assets and addressing design or engine issues rapidly » (Arnaud 2011, 25).
446. « Unreal Editor (UnrealEd) is a pure "What You See is What You Get" content creation tool » (Epic Games 2005).
447. any adjustment to the original base game, whether a tiny tweak or a complete re-imagining » (Dyer 2016).

d'engagement diffère, allant d'utilisateurs de mods à créateurs de mods » (Bilińska-Reformat, Dewalska-Opitek et Hofman-Kohlmeyer 2020, 1 TDR)[448], où certains *mods* sont éminemment techniques et d'autres relativement simples. Cependant, il s'agit d'une altération d'un jeu existant pour Scacchi, où « des jeux modifiés ne sont pas des *standalones*, puisqu'ils nécessitent que l'utilisateur possède une copie originale, acquise légalement ou via une licence, du logiciel de jeu non modifié » (Scacchi 2011, 62 TDR)[449]. De surcroît, nous estimons que les méthodes de développement d'un jeu vidéo sont substantiellement différentes des méthodes pour la conception d'un *mod*. L'objectif principal de notre thèse est bien de voir la création des mécanismes interactifs avec les véritables outils de développement, et non pas via des outils de *modding*.

Dans tous les cas, nous avons observé que les deux outils principaux, *RenderWare* et *Unreal Engine 2*, étaient majoritairement utilisés pour développer des jeux appartenant à de multiples genres vidéoludiques. C'est pour cette raison que nous éludons des listes qui recensent les multiples jeux, et genres vidéoludiques associés, pour les moteurs les plus importants. Nous considérons, en effet, que cette information n'est pas pertinente au regard de la diversité des genres de jeux développés via chaque moteur de jeu. De telles listes ne pourront nullement fournir des informations utiles ou exploitables dans le cadre de notre recherche. Néanmoins, dans un tel cas, l'affirmation de Gregory pourrait sembler paradoxale, puisqu'il indique que « chaque genre de jeu a ses propres exigences technologiques particulières (...) [et] les moteurs de jeux ont traditionnellement présenté des différences significatives d'un genre à un autre » (Gregory 2019, 31 TDR)[450]. Ce constat est également mis en exergue par Young, qui indique que « la plupart des moteurs de jeu se focalisent sur un genre particulier » (Young 2005, 1 TDR)[451].

Toutefois, ce que Young et Gregory ont constaté, ce n'est pas tant une incompatibilité des moteurs de jeu à être utilisés pour d'autres genres que celui

448. « customers engage in game modding, but the level of engagement differs, from mod users to mod creators » (Bilińska-Reformat, Dewalska-Opitek et Hofman-Kohlmeyer 2020, 1).
449. « modded games are not standalone systems, as they require the user to have an originally acquired or licensed copy of the unmodded game software. » (Scacchi 2011, 62).
450. « each game genre has its own particular technological requirements (...) [and] game engines have traditionally differed quite a bit from genre to genre » (Gregory 2019, 31).
451. « most game engines focus on a particular genre » (Young 2005, 1).

pour lequel ils ont été conçus. Il s'agit plutôt de considérer qu'un jeu développeur sur un moteur conçu pour un autre genre risque de manquer d'optimisation, parfois de manière substantielle. C'est pourquoi Young précise bien que ces moteurs « sont mieux adaptés au type de jeux pour lesquels ils ont été conçus » (Young 2005, 1 TDR)[452]. Cette même notion d'optimisation pour les moteurs avait de nombreuses conséquences, en particulier pour la sixième génération de consoles.

Ainsi, elle impliquait de faire la distinction entre deux types de moteurs de jeu : les *indoor engines* et les *outdoor engines*. Nous pourrions traduire ces termes par les moteurs d'intérieur et moteurs d'extérieur, sans toutefois utiliser ces traductions pour les raisons évoquées en introduction. Gregory indique sur ce point, et particulièrement concernant les FPS, que « la technologie de rendu utilisée (...) est presque toujours très optimisée (...) pour un type particulier d'environnement rendu » (Gregory 2019, 15 TDR)[453], renforçant cette idée d'optimisation pour les moteurs. Il évoque ainsi plusieurs algorithmes, notamment pour des « jeux vidéo (...) en intérieur [qui] utilisent souvent des arbres *Binary Space Partitioning* ou des systèmes de rendu basés sur les portails [tandis que] les jeux (...) en extérieur utilisent d'autres types d'optimisations de rendu » (Gregory 2019, 15 TDR)[454].

Nous pouvons également illustrer ce point à travers le moteur du jeu *Metal Gear Solid 2: Sons of Liberty* (Konami 2001, PS2), hypothétiquement spécialisé *indoor* puisque, selon Hivner, le « directeur Hideo Kojima explique que les jeux précédents de la série se déroulaient en intérieur simplement parce que la technologie (...) n'était pas assez puissante pour représenter une véritable jungle » (Hivner 2003 TDR)[455]. C'est pourquoi une adaptation du moteur en environnement extérieur était nécessaire pour le développement de *Metal Gear Solid 3: Snake Eater* (Konami 2004, PS2). Ainsi, Hivner rajoute que, « afin de

452. « are best suited to the type of games they were designed for » (Young 2005, 1).
453. « the rendering technology employed (...) is almost always highly optimized (...) to a particular type of environment being rendered » (Gregory 2019, 15).
454. « indoor (...) games [which] often employ binary space partitioning trees or portal-based rendering systems [while] outdoor (...) games use other kinds of rendering optimizations » (Gregory 2019, 15).
455. « director Hideo Kojima explains that previous games in the series stayed indoors simply because the technology (...) weren't powerful enough to portray a true jungle » (Hivner 2003).

fournir un tel visuel (…), le moteur de MGS2 a dû être abandonné, [et] pour Snake Eater, l'équipe a développé un tout nouveau moteur *from scratch* » (Hivner 2003 TDR)[456].

Au regard de tous ces éléments, nous avons constaté qu'il y a eu une grande quantité de moteurs de jeu développés et commercialisés lors de la sixième génération de consoles. Il semble également que le moteur de jeu *Unreal Engine 2* et les deux version de *RenderWare* soient les outils les plus populaires de cette génération. Néanmoins, il existait encore une segmentation importante entre les professionnels et les indépendants, voire les amateurs. C'est pourquoi nous allons passer en revue, dans la prochaine sous-section, le développement de jeux vidéo via différents outils de création disponibles pour le grand public.

456. « in order to provide such visual (…), MGS2's engine had to go, [and] for Snake Eater, the team built a brand-new engine from scratch » (Hivner 2003).

4-1-C. Démocratisation des outils indépendants

Lors de la sixième génération de consoles, nous pouvons observer une démocratisation de plusieurs solutions et outils grand public pour le développement d'un jeu vidéo amateur, notamment des moteurs de jeu propriétaires pour indépendants et des moteurs de jeu *Open Source*. Néanmoins, avant d'évoquer ces solutions, on constate que le développement de jeux vidéo sur les consoles de salon était encore relativement fermé. Pourtant, il existait bien deux outils disponibles sur la *PlayStation 2* : le *Linux Kit* ainsi que le langage de programmation *YA-Basic*. Nous allons justement mettre en surbrillance ces deux solutions pour en vérifier la potentialité dans le développement de jeux.

Au regard du *Linux Kit*, il s'agit d'un outil de développement composé, comme le rappelle Christophe Guillemin, d'un « disque dur d'une capacité de 40 Go, à installer dans la console, d'un clavier et d'une souris » (Guillemin 2002). Surtout, son objectif principal est d'abord de pouvoir « transformer (...) [une] console Playstation 2 en ordinateur de bureau » (*ibid.*), via le système d'exploitation *Linux*. Mais il est possible de citer une recherche allemande de Fackroth *et al.*, où ils ont développé « un jeu d'action arcade en 3D, *Magnetic Masters*, sur la Playstation 2 en utilisant uniquement le Linux Kit de la PS2 » (Fackroth *et al.* 2005, 1 TDR)[457]. Cela permettait de tester le potentiel du *Linux Kit* dans le cadre de la création d'un jeu vidéo, ce qui a abouti à la création de leur « moteur de jeu, qui possédait un système graphique (...) capable de faire un rendu de scènes avec jusqu'à 30 000 polygones texturés (...) à 30 FPS » (Fackroth *et al.* 2005, 1 TDR)[458].

Le résultat de cette recherche allemande permet effectivement de démontrer la possibilité d'utiliser le *Linux Kit* dans le cadre du développement d'un jeu, de manière similaire au kit de développement *Net Yaroze* de la génération précédente. Néanmoins, et également à l'instar de la *Net Yaroze*, la conclusion de cette recherche a insisté sur le fait que le « développement de jeux en utilisant le Linux Kit de la PS2 est possible – mais avec de nombreuses restrictions et pas

457. « a 3D action arcade game, *Magnetic Masters*, on the Playstation 2 using solely the PS2 Linux Kit » (Fackroth *et al.* 2005, 1).
458. « game engine, that features a graphics system (...) able to render scenes with up to 30.000 polygons with textures (...) at 30 fps » (Fackroth *et al.* 2005, 1).

de manière aussi élaborée que pour des jeux PS2 professionnels » (Fackroth *et al*. 2005, 6 TDR)[459].

Le second outil que nous avons mentionné est, pour reprendre le descriptif de Tony Smith, le « YA-Basic, une implémentation open source du langage Basique' de programmation » (Smith 2000 TDR)[460]. Néanmoins, il apparaît que le *YA-Basic* ne constitue pas un véritable outil de développement de jeux vidéo, mais plutôt une simple démonstration technologique, sans réelle application. Plus encore, le contexte de publication de ce langage dérivé du *BASIC* pourrait être lié, selon une hypothèse indiquée par Smith, à une taxe puisque « le bundle YA-Basic est exclusif à l'Europe, le seul territoire à imposer un tarif d'importation différent sur les ordinateurs et le matériel de jeux vidéo » (Smith 2000 TDR)[461]. En revanche, nous ne pouvons pas nous prononcer sur la véracité d'une telle hypothèse.

Dans tous les cas, nous constatons que le développement de jeux vidéo dit professionnel sur la *PlayStation 2*, du moins au regard de la section précédente avec les exemples comme *Unreal Engine 2* ou *RenderWare*, semblait être réservé à des développeurs possédant des licences. Les deux outils, le *Linux Kit* et le *YA-Basic* sur la PS2, ne permettent aucunement de réaliser des jeux aussi complexes que les outils professionnels. En revanche, il existait pléthore de moteurs de jeu sur PC, que nous pouvons subdiviser en deux catégories : les moteurs de jeu propriétaires pour indépendants et les moteurs de jeu *Open Source*. En apparence, on pourrait signaler qu'on s'éloigne de notre cadre initial d'étudier uniquement le développement de jeux sur consoles. Or, nous verrons dans les prochains chapitres que la compréhension de certains outils *PC* est capitale, car celle-ci va *in fine* influencer le développement sur les consoles.

Ainsi, nous pouvons constater qu'il existe de nombreux moteurs de jeu propriétaires pour indépendants, mais notre objectif est de centrer notre propos sur *DarkBasic Pro*. Celui-ci utilisait une variante du langage *Basic* pour le

459. « game development using the PS2 Linux Kit is possible – but with many restrictions and not to the extend of professional PS2 games » (Fackroth *et al*. 2005, 6).

460 . « YA-Basic, an open source implementation of the classic beginners' programming language » (Smith 2000).
Note : il y a un jeu de mot dans la citation originale avec *Beginners'*, qui fait en réalité référence à la première lettre de l'acronyme BASIC. Nous avons tenté de reproduire ce jeu de mot avec *Basique*.

461. « the YA-Basic bundle is exclusive to Europe, the only territory to impose a different import tariff on computers and game equipment » (Smith 2000).

développement de jeux vidéo, nommée *DarkBasic*, et spécialement conçue pour ce même logiciel de création sur PC. Plus précisément, et comme Jonathan Harbour et Joshua Smith peuvent le préciser, il s'agit d'un « langage de programmation qui ressemble à *Microsoft QuickBasic*, un descendant du langage BASIC original (*Beginner's All-Purpose Symbolic Instruction Code*) » (Harbour et Smith 2003, 8 TDR)[462]. Ce logiciel, s'arc-boutant donc sur du *BASIC*, semble pourtant être un « moteur de jeu scripté presque au même niveau que les moteurs de jeu commerciaux » (*ibid.*, 6 TDR)[463] de l'époque. Même s'il est nécessaire de constater, dans un tel cadre, que « DarkBASIC n'est pas optimisé comme (...) d'autres moteurs pour un but spécifique (comme les jeux de tir à la première personne) ; au contraire, il offre un moteur de jeu polyvalent pour plusieurs genres » (*ibid.* TDR)[464].

Pour autant, les technologies différentes exploitées par le moteur *DarkBasic* semble être limitatives, puisque le langage « DarkBASIC a été conçu pour la création de jeux, il ne possède pas la complexité de langages comme le (...) C++ (le langage le plus utilisé pour les jeux commerciaux) » (*ibid.* TDR)[465]. Néanmoins, l'absence de complexité ne pourrait être liée simplement au fait qu'il s'agit d'un langage spécifiquement conçu pour créer des interactions ou du *gameplay*. Nous pourrions supputer cela plutôt à la volonté de simplifier la création et l'utilisation du langage en question. Néanmoins, il est indiqué que, « bien que vous puissiez (...) écrire (...) des jeux avec DarkBASIC, vous pouvez également l'utiliser pour rapidement prototyper un jeu que vous prévoyez de programmer dans un langage plus difficile ultérieurement » (*ibid.*, 14 TDR)[466]. En clair, le moteur *DarkBasic Pro* semble être un outil de création possédant certaines contraintes, principalement

462. « programming language resemble Microsoft QuickBasic, a descendant of the original BASIC (Beginner's All-Purpose Symbolic Instruction Code) language » (Harbour et Smith 2003, 8).
463. « scripted game engine somewhat at the level of commercial game engines » (Harbour et Smith 2003, 6).
464. « DarkBASIC is not optimized like these other engines for a specific purpose (such as first-person shooters); rather, it provides a general-purpose game engine for multiple genres » (Harbour et Smith 2003, 6).
465. « DarkBASIC was designed for writing games, it contains none of the complexity of languages such as (...) C++ (the language most commonly used for commercial games") » (Harbour et Smith 2003, 6).
466. « although you *can* (...) write (...) games with DarkBASIC, you could also use it to quickly prototype a game that you plan to eventually write in a more difficult language » (Harbour et Smith 2003, 14).

liées à une absence de performance et d'optimisation au regard des moteurs commerciaux comme *Unreal Engine 2*.

Et, en ce qui concerne les moteurs *Open Source* publiquement disponibles dans la sixième génération, voire la septième, on peut constater, selon les informations de Peternier, Vexo et Thalmann, que « Ogre, Crystal Space et Irrlicht figurent parmi les moteurs graphiques open source les plus utilisés et complets disponibles jusqu'à présent » (Peternier, Vexo et Thalmann 2008, 243 TDR)[467]. Il est néanmoins question de moteur graphique, qu'on peut également appeler un moteur de rendu 3D, c'est-à-dire un composant qui s'occupe simplement de gérer l'affichage graphique d'un logiciel. Celui-ci est substantiellement différent de la définition d'un moteur de jeu, puisque Gregory va définir le moteur graphique comme étant « principalement préoccupé par le rendu de surface » (Gregory 2019, 624 TDR)[468]. Il précise également que les « développeurs de jeux ont traditionnellement modélisé leurs surfaces en utilisant des maillages triangulaires » (*ibid.*, 625 TDR)[469]. Cela signifie concrètement qu'il n'est pas possible de créer des mécanismes interactifs avec un moteur graphique, et que ces multiples outils sont totalement éloignés de notre recherche. Naturellement, il existait d'autres moteurs de jeu à cette époque, mais l'objectif de cette recherche est surtout de se focaliser sur les moteurs majeurs utilisés pour des projets commerciaux. Désormais, au regard de toutes les informations accumulées, nous pouvons passer à l'étape suivante pour élaborer des théories sémio-informatiques, et cela dès la prochaine sous-section.

467. « Ogre, Crystal Space and Irrlicht are among the most used and complete open-source graphics engines available so far » (Peternier, Vexo et Thalmann 2008, 243).
468. « primarily concerned with rendering *surfaces* » (Gregory 2019, 624).
469. « game deveopers have traditionally modeled their surfaces using triangle meshes » (Gregory 2019, 625).

4-2. CARACTÈRES PRIMORDIAUX INTERACTIFS

À l'aune de la situation du développement de jeux dans la sixième génération, les différentes technologies informatiques jouent un rôle prépondérant pour la création de l'interactivité. L'introduction du langage *C++* puis l'émergence, voire la première démocratisation massive, des moteurs de jeu professionnels permet désormais de complexifier notre signe interactif à travers l'élaboration de trois caractéristiques majeures. Les deux premières caractéristiques reprennent les concepts du signe linguistique exposé dans le CLG, lequel mentionnant que le « signe linguistique (...) possède deux caractères primordiaux » (Saussure 2005, 75), qui sont l'arbitraire et la linéarité du signe linguistique. Surtout, l'intérêt principal serait de juxtaposer ces deux concepts aux caractéristiques des langages de programmation, ici le *C++.* Néanmoins, la troisième caractéristique va plutôt considérer un autre aspect majeur, celui de la complexité des instructions, concept lié à des intérêts informatiques et ayant des conséquences importantes sur la conception de l'interactivité. Ainsi, nous pouvons présenter les trois concepts principaux pour complexifier les signes interactifs, ce que nous nommons les caractères primordiaux de l'interactivité :

- Caractère arbitraire ;
- Caractère de linéarité apparente ;
- Caractère de complexité des instructions.

Ainsi, le caractère arbitraire du signe interactif haut niveau peut être conceptualisé par le fait que « le lien unissant le signifiant au signifié est arbitraire » (Saussure 2005, 75). Dans notre cas, cela signifie qu'il n'existe aucun lien naturel entre le signifiant virtuel et le signifié virtuel. Autrement dit, le lien « est immotivé (...) par rapport au signifié, avec lequel il n'a aucune attache naturelle dans la réalité » (*ibid.*, 76). La première conséquence de l'arbitraire de notre signe interactif est qu'il est possible de remplacer un signifiant par un autre, tout en conservant un signifié identique.

Par exemple, nous pouvons analyser le signe interactif pour entrer à l'intérieur d'un véhicule dans *Grand Theft Auto III* (Rockstar Games 2001, PS2). Dans ce cas, *l'input* qui déclenche la réaction est différent en fonction de la version *PlayStation 2*, *Xbox* ou *PC* du jeu. Nous pouvons également signaler que la version PC permet de redéfinir les *inputs*, ce qui démontre bien un aspect

arbitraire du signifiant. Nous avons naturellement conscience, pour reprendre les propos de Chandler, que « de nombreux critiques ont trouvé le principe de l'arbitraire de Saussure difficile, voire impossible, à accepter » (Chandler 2022, 21 TDR) [470]. Néanmoins, nous estimons que ce même principe de l'arbitraire saussurien correspond bel et bien à notre signe interactif haut niveau. De surcroît, et comme mentionné en introduction, nous n'entrons pas dans ces débats conceptuels. D'autant plus que l'usage des autres périphériques peut renforcer cette arbitraire du signe, notamment la caméra ou le casque de réalité virtuelle, mais cela est hors du cadre de notre recherche.

Nous pouvons à présent nous intéresser au caractère de linéarité apparente du signe interactif haut niveau. Nous définissons cette linéarité en nous appuyant sur les propos du CLG, c'est-à-dire comme étant le fait qu'un « signifiant (...) se déroule dans le temps seul » (Saussure 2005, 77). Néanmoins, l'utilisation du terme *apparent* pour ce caractère nous permet de préciser que le concept de linéarité possède une acception très précise dans notre cas, et porte surtout sur l'action puis la réaction d'un seul et unique signe interactif. De surcroît, et pour reprendre les propos de Gregoire, nous soulignons que, dans la programmation des « applications larges et modernes, il est rare qu'il y ait une séquence linéaire d'événements, [et] souvent, un utilisateur est capable d'exécuter n'importe quelle commande à tout moment » (Gregoire 2021, 170 TDR)[471]. De plus, il existe des techniques spécifiques pour exécuter un code de manière non linéaires en programmation, notamment à travers ce qu'il est possible d'appeler le *multithreading*. Néanmoins, une telle précision technique n'est pas pertinente au regard du module de précision technologique de notre approche du DLT, et nous éludons ainsi celle-ci.

Au-delà de l'aspect technologique non linéaire, on pourrait également objecter que l'interactivité elle-même n'est pas nécessairement linéaire, et que les mondes ouverts dans lesquels des interactants évoluent ne sont aucunement linéaires. Par exemple, plusieurs signes interactifs peuvent s'exécuter en même

470. « many critics have found Saussure's principle of arbitrariness difficult if not impossible to accept » (Chandler 2022, 21).

471. « large, modern applications, there is rarely a linear sequence of events, [and] often a user is able to perform any command at any time » (Gregoire 2021, 170).

temps. Certes, mais lorsque nous isolons un seul et unique signe interactif haut niveau, c'est-à-dire une suite d'action(s) et de réaction(s), nous analysons celui-ci comme ayant une exécution séquentielle et linéaire, même si cela entre en contradiction avec l'exécution technique. L'objectif étant de pouvoir décomposer celui-ci de manière limpide.

Puis, au regard du caractère de complexité des instructions, nous pouvons définir celle-ci comme étant la mesure du nombre d'instructions nécessaires pour effectuer une interaction, ce qui pourrait être utile pour pouvoir comparer les signes interactifs entre eux. Par exemple, l'ouverture d'une porte ou le fait de réaliser une translation d'un objet via un interrupteur représente deux signes interactifs. Néanmoins, ceux-ci peuvent être plus ou moins complexes, et représenter une quantité variable d'instructions, aussi bien dans un langage textuel comme le *C++* que dans un langage visuel comme le *Blueprint*. Ce troisième caractère permet surtout de mettre en exergue le niveau de langage utilisé.

Nous pouvons surtout mentionner, grâce à l'étude de Brooks, que « la méthode la plus efficace pour la productivité, la fiabilité et la simplicité du développement logiciel a été l'utilisation progressive de langages haut niveau pour la programmation » (Brooks 1987, 13 TDR)[472]. Nous pouvons ainsi affirmer que le *C* puis le *C++* représentaient ainsi deux langages haut niveau, du moins par rapport à *l'assembleur*, et augmentaient ainsi hypothétiquement la productivité d'une manière substantielle. Néanmoins, nous pouvons également évoquer le concept nommé pénalité d'abstraction en informatique, ou *abstraction penalty* en anglais. Il est possible de définir celui-ci comme étant, selon Williams, « les coûts d'implémentation associés à l'utilisation de fonctionnalités haut niveau, en comparatif à l'utilisation directe de fonctionnalités bas niveau sous-jacentes » (Williams 2019, 12 TDR)[473]. C'est pourquoi cette notion de pénalité d'abstraction est la fondation théorique de ce troisième critère primordial de complexité des instructions. Et nous pouvons étudier, à travers le précepte de comparaison

472. « the most powerful stroke for software productivity, reliability, and simplicity has been the progressive use of high-level languages for programming » (Brooks 1987, 13).

473. « the implementation costs associated with using any high-level facilities, compared to using the underlying low-level facilities directly » (Williams 2019, 12).

anachronique, le nombre d'instructions à la fois en *Blueprints* ainsi qu'en *C++* sur *Unreal Engine 4* pour réaliser de tels signes interactifs.

Enfin, on peut constater que les aspects haut niveau et bas niveau des instructions, par exemple entre les *Blueprints* et le *C++* sur *Unreal Engine 4*, est également similaire aux concepts évoqués par Courtés des « structures de surface et les structures profondes en linguistique et, plus largement, en sémiotique » (Courtés 1991, ch. 1.2). Néanmoins, ces implications pour des structures de surface et des structures profondes peuvent également être relativement proches du prochain concept que nous allons développer, celui d'une interactivité et de son rapport avec les générations de consoles.

4-2-A. INTERACTIVITÉ GÉNÉRATIONNELLE

Au-delà des trois caractères primordiaux du signe interactif, nous pouvons désormais développer une théorie sémio-informatique pour mieux appréhender les propriétés d'un signe interactif, et le contrôle que peut avoir l'interactant pour potentiellement altérer le signe en question. Une même théorie que nous nommons explicitement interactivité générationnelle, et qui permet de mettre en exergue un autre phénomène tout aussi important pour les signes interactifs bas niveau : l'évolution des interactions dans une même génération, mais également à travers les multiples générations.

Afin d'élaborer une telle théorie, il est d'abord nécessaire de reprendre deux concepts du signe linguistique du CLG, autrement dit « l'immutabilité et la mutabilité du signe » (Saussure 2005, 82). Cela permettra à la fois de vérifier la stabilité des signes interactifs dans une génération donnée, mais également l'instabilité de ceux-ci à travers plusieurs générations. De surcroît, l'intérêt est de développer ces concepts pour mieux appréhender les limitations dans les moteurs de jeu, et plus particulièrement pour les instructions de programmation, autrement dit la substance développementale du plan du contenu du signe interactif bas niveau. Précisons également que nos notions de mutabilité et d'immutabilité concernent à la fois le signifiant et le signifié. Ainsi, les trois concepts que nous allons développer sont donc les suivants :

- Immutabilité interactive ;
- Mutabilité interactive ;
- Ancrage générationnel.

Ainsi, nous pouvons d'abord développer le concept de l'immutabilité interactive en s'appuyant sur le CLG, indiquant que « le signe est immuable, c'est-à-dire résiste à toute substitution arbitraire » (Saussure 2005, 80). Cette immutabilité est induite par le caractère arbitraire du signe, puisque « c'est parce que le signe est arbitraire qu'il ne connaît d'autre loi que celle de la tradition, et c'est parce qu'il se fonde sur la tradition qu'il peut être arbitraire » (Saussure 2005, 82). Dans notre cas, nous pouvons définir l'immutabilité interactive comme étant l'exécution d'un signe interactif haut niveau d'une manière stable et sans changement possible, sans tenir compte des variations interactives résiduelles, concept exposé dans le deuxième chapitre. Cependant, cette immutabilité

s'inscrit dans les deux versions de notre signe interactif. Pour notre signe interactif haut niveau, l'immutabilité symbolise simplement le fait d'avoir des constantes à la fois pour le signifiant virtuel et le signifié virtuel, sans possibilité de les modifier. Cela signifie que les actions et réactions seront toujours identiques, sans aucune altération possible autre que des variations interactives résiduelles.

En revanche, pour évoquer la notion d'immutabilité interactive pour le signe interactif bas niveau, il est nécessaire de comparer ce concept avec celui de l'immutabilité en *C++* pour vérifier comment le langage peut le mettre en œuvre. Il s'agit notamment de ce qui est nommé *const correctness*, qu'Horton et Van Weert vont définir comme étant la possibilité, « pour des objets *const*, (...) [de] seulement appeler les fonctions membres *const* (...) [et] les objets *const* doivent être totalement immutable » (Horton et Van Weert 2020, 425 TDR) [474] . Néanmoins, cela signifie qu'un objet est totalement immutable, et qu'aucun changement n'est possible, sauf dans un seul cas que nous aborderons plus loin (la mutabilité). De surcroît, ils précisent que « les restrictions imposées par le compilateur sont appelées *const correctness* — cela empêche les objets *const* d'être mutés » (Horton et Van Weert 2020, 425 TDR)[475], et que le compilateur va donc empêcher une quelconque modification d'un objet *const*.

Cela signifie que l'immutabilité en *C++* entre en contradiction avec notre concept de l'immutabilité interactive, et n'est pas applicable pour notre signe interactif. De surcroît, la définition même de notre signe interactif implique obligatoirement un changement d'état de l'univers virtuel, ce qui n'est pas envisageable avec l'immutabilité du *C++.* En revanche, si l'immutabilité interactive, contrairement à l'immutabilité en *C++,* implique un changement d'état, elle considère également que l'interactant ne pourrait aucunement altérer l'interaction. S'il existe des possibilités pouvant altérer le signe interactif mais qu'elles ne sont pas contrôlables par le joueur, par exemple le climat de *Grand Theft Auto III* (Rockstar Games 2001, PS2), alors il s'agit également de variations

474. « for const objects, (...) [to] only call const member functions (...) [and] const objects must be totally immutable » (Horton et Van Weert 2020, 425).

475. « compiler-enforced restrictions is called const correctness—it prevents const objects from being mutated » (Horton et Van Weert 2020, 425).

interactives résiduelles. Nous pouvons désormais clarifier le concept de l'immutabilité interactive à travers le tableau ci-dessous, permettant ainsi d'éviter les potentielles confusions, bien que nous reconnaissions que la polysémie du terme immutabilité, dans notre étude, complexifie la situation.

Notion	Définition
Immutabilité interactive haut niveau	Action ou ensemble d'actions qui entraînent des réactions identiques, sans altération possible
Immutabilité interactive bas niveau	Exécution identique d'un script *C++* ou *Blueprints*, sans variables (ou événement, etc.) permettant une altération des actions et réactions
Immutabilité C++	Fonction membre constant ou objet constant ne permettant aucune modification ni altération

Tableau 28. Immutabilité interactive

Au regard de la mutabilité interactive, il est possible de définir celle-ci en exposant un principe général, comme ce fut le cas pour le CLG, indiquant que « le temps altère toutes choses ; il n'y a pas de raison pour que la langue échappe à cette loi universelle » (Saussure 2005, 85). De plus, on peut également considérer avec le CLG que « la continuité du signe dans le temps, lié à l'altération dans le temps, est un principe de la sémiologie générale » (Saussure 2005, 84). Ainsi, nous pouvons nous inspirer de cela pour définir que la mutabilité interactive est la possibilité, pour un ensemble d'actions et des réactions, de subir des altérations par l'interactant. Et, tout comme l'immutabilité, notre notion de la mutabilité interactive se subdivise en deux : la mutabilité d'un signe interactif haut niveau, qui concerne surtout les possibilités de modification qu'on observe dans le jeu, et la mutabilité d'un signe interactif bas niveau, avec les éléments techniques qui indiquent les altérations possibles, l'exemple le plus simple étant celui des variables.

Nous pouvons préciser que, tout comme l'immutabilité, la mutabilité existe également dans le langage *C++*. Il s'agit, pour reprendre la définition de Dawson, de « membres de données qui peuvent être modifiés même par des fonctions

membres constantes » (Dawson 2014, 298 TDR)[476]. Ainsi, en utilisant le mot-clef *mutable* à une propriété, il est possible de la modifier même en utilisant une méthode constante, aussi nommée fonction membre constante en dehors d'une classe. Et, ce qui pourrait être intéressant à noter, c'est que la mutabilité en *C++* permet de modifier une valeur, même pour une méthode ou une fonction membre constante. Néanmoins, cela ne correspond toujours pas à notre notion de la mutabilité interactive et de l'immutabilité interactive.

Enfin, la dernière notion que nous souhaitons développer dans cette sous-section permet de spécifier l'existence de la mutabilité et de l'immutabilité d'un signe interactif, afin de les mettre en corrélation soit avec plusieurs jeux d'une génération, soit entre plusieurs générations. On parlerait d'une présence d'ancrage pour valider celle-ci ou d'une absence d'ancrage dans le cas contraire. Surtout, nous subdivisons l'ancrage générationnel en deux parties : un ancrage intragénérationnel, c'est-à-dire une mutabilité ou immutabilité entre plusieurs jeux d'une même génération, ainsi qu'un ancrage intergénérationnel, autrement dit entre plusieurs générations technologiques différentes. Nous spécifions également que l'ancrage ne nécessite pas de comparer des jeux vidéo proches ou des suites, mais simplement des signes interactifs similaires. Il est ainsi possible de synthétiser ces trois aspects de l'interactivité générationnelle dans le tableau ci-dessous.

476. « data members that can be modified even by constant member functions » (Dawson 2014, 298).

Notion	Définition
Mutabilité interactive	Action ou ensemble d'actions qui entraînent des réactions pouvant être altérées par un interactant
Immutabilité interactive	Action ou ensemble d'actions qui entraînent des réactions identiques, sans altération possible
Ancrage intragénérationnel	Signes interactifs présents dans plusieurs jeux différents d'une même génération, partageant des propriétés mutables ou immutables similaires
Ancrage intergénérationnel	Signes interactifs présents dans des jeux de multiples générations, partageant des propriétés mutables ou immutables similaires

Tableau 29. Synthèse de mutabilité, immutabilité et ancrage générationnel

Désormais, il est nécessaire d'illustrer ces multiples concepts à travers notre corpus vidéoludique. Ainsi, nous pouvons mettre en exergue l'immutabilité d'un signe interactif haut niveau via le jeu *Grand Theft Auto III* (Rockstar Games 2001, PS2), où l'interactant humanoïde Claude peut conduire un véhicule comme la *Diablo Stallion*. Ce signe interactif est néanmoins immutable parce qu'il n'est pas possible de modifier ou d'altérer la manière de conduite, c'est-à-dire les multiples actions ou réactions pendant la conduite. En d'autres termes, il n'existe pas de variables ou d'événements contrôlables qui peuvent modifier le comportement d'une supposée classe conduite, du moins pour un même véhicule. En revanche, si nous analysons le jeu *Grand Theft Auto: San Andreas* (Rockstar Games 2004, PS2), le signe interactif de conduite est mutable. En effet, il existe des compétences de conduite de l'interactant permettant de modifier *in fine* le comportement du véhicule et la manière de le conduire, c'est-à-dire la réaction du signe interactif. Dans ce même exemple, il y a une absence d'ancrage intergénérationnel de l'immutabilité, puisque le signe interactif conduire est immutable dans *Grand Theft Auto III* (Rockstar Games 2001, PS2) mais mutable pour *Grand Theft Auto: San Andreas* (Rockstar Games 2004, PS2). Ces multiples exemples permettent de mettre en exergue le concept des mondes ouvert, qui est justement l'objet de notre prochaine sous-section.

4-2-B. DEGRÉ D'AGNOSTICISME DÉVELOPPEMENTAL

Au-delà des perspectives développées par les multiples caractères primordiaux interactifs de notre signe interactif, nous pouvons également évoquer la disponibilité publique, ou non, des différentes technologies. Et, à ce titre, notre objectif est d'élaborer une théorie pouvant rendre compte des différences liées à la disponibilité ou l'indisponibilité d'une technologie comme un moteur de jeu. L'intérêt d'une telle théorie est de vérifier les éventuelles conséquences sur la création des mécanismes interactifs, ainsi que l'analyse d'un tel processus. Celle-ci se base sur la notion informatique d'agnosticisme exposée par Gavin Wright, un « terme [qui] peut faire référence à des logiciels ou du matériel, ainsi qu'à des processus ou pratiques commerciales » (Wright 2022 TDR) [477] . Celui-ci implique surtout une absence de connaissance du fonctionnement des technologies (*ibid.*). C'est grâce à ce concept que nous développons le concept du degré d'agnosticisme, en formulant trois degrés différents :

- Degré d'agnosticisme 0 ou technologie transparente ;
- Degré d'agnosticisme 1 ou technologie semi-transparente ;
- Degré d'agnosticisme 2 ou technologie opaque.

Tout d'abord, nous définissons le degré d'agnosticisme zéro, ou celui d'une technologie transparente, comme étant disponible pour le grand public, mais possédant également un code source public. Un degré impliquant la disponibilité intégrale du code source d'un moteur donné, et non pas une version dont le code source est partiellement disponible. Nous avons ainsi constaté une disponibilité du code source complet de plusieurs moteurs et outils de développement de jeux vidéo. Par exemple, les moteurs graphiques *Irrlicht* ou *OGRE* sont des moteurs ayant un degré d'agnosticisme 0, puisque le code source est intégralement disponible. La disponibilité du code source permet surtout d'avoir une compréhension plus importante du fonctionnement interne du moteur. Ainsi, pour utiliser notre outil de comparaison anachronique du DLT, nous pouvons constater que le code source du moteur *Unreal Engine 4* est intégralement

477 . « term [that] can refer to software and hardware, as well as business processes or practices » (Wright 2022).

disponible dans la sphère publique. Cela catégorise d'emblée ce dernier moteur comme étant une technologie transparente, et ainsi plus aisément compréhensible. C'est pourquoi celui-ci peut être étudié d'une manière très approfondie, aussi précisément que nécessaire, afin d'analyser cette technologie sans fard ni obscurité. Toujours en utilisant la comparaison anachronique, nous pourrions également citer d'autres moteurs dont le code source est disponible, notamment *Godot*, devenant parfaitement adaptés à une étude scientifique. Il s'agit donc d'un avantage substantiel lors d'une analyse, puisqu'il n'y a pas d'informations cachées. Naturellement, nous éludons les questions relatives à la licence du code source, puisque les statuts *Open Source* ou *Source Available*, respectivement pour *Godot* et *Unreal Engine*, concernent seulement le besoin de modification d'un moteur et non pas son analyse.

Ensuite, au regard du deuxième niveau de cryptage, on va définir le degré d'agnosticisme 1, ou une technologie semi-transparente, comme étant une technologie publiquement disponible, mais dont le code source est indisponible. On précise que la disponibilité de cette même technologie doit signifier une possibilité d'utiliser la totalité des caractéristiques de cette technologie, de manière illimitée dans le temps, voire dans l'espace géographique. L'exemple pouvant le mieux illustrer ce cas de figure est celui du moteur de jeu *Unity*, disponible pour le grand public mais dont le code source est indisponible. Certes, le « plan Unity Entreprise fournit (...) [aux développeurs] l'accès au code source » (Unity Technologies s. d.-b TDR)[478], mais il ne s'agit pas d'une disponibilité publique, qui ne pourrait donc pas être exploitable dans un cadre scientifique. Certes, nous pouvons estimer qu'une technologie semi-transparente pourrait être utilisable dans une étude scientifique. Néanmoins, celle-ci possède des limites inhérentes, impliquant qu'une partie du fonctionnement de cette technologie resterait *de facto* inconnue.

Enfin, nous pouvons évoquer le troisième niveau, celui d'un degré d'agnosticisme 2, ou d'une technologie opaque, indiquant qu'il n'existe aucune possibilité d'accéder publiquement à une version de cette technologie.

478 . « Unity Enterprise plan provides (...) [developers] with Source Code Access » (Unity Technologies s. d.-b).

Naturellement, et au regard du cadre scientifique de notre étude, notre notion de technologie opaque ne différencie aucunement les moteurs internes des moteurs propriétaires pouvant être acquis via une licence commerciale, comme ce fut le cas pour l'API *RenderWare Platform*. Car, dans les deux cas, les informations ne peuvent ni être publiées, ni exploitables en dehors de la licence d'achat. Nous considérons donc qu'à partir du moment où une technologie est indisponible publiquement, celle-ci devient intégralement opaque.

Cette théorie du degré d'agnosticisme développemental pourrait sembler évidente. Néanmoins, sa formulation explicite possède des conséquences fondamentales sur l'analyse des différents outils de développement ainsi que sur notre approche du DLT. Tout d'abord, cela implique qu'il est difficile de réaliser une étude portant sur le fonctionnement des technologies opaques. Par exemple, étudier des moteurs indisponibles pour le grand public ou des API commerciales est une tâche difficilement intégrable dans un cadre scientifique. Un phénomène que nous avons constaté avec l'analyse des moteurs classés par Pedersen (2003), de l'API *RenderWare Platform* ainsi que du moteur *Unreal Engine 2*. Les informations semblaient être parcellaires mais surtout invérifiables. Naturellement, cela exclut d'emblée une analyse scientifique des moteurs de jeu propriétaires et les moteurs de jeu faits maison, sauf si cette analyse est purement basée sur des aspects discursifs de l'industrie vidéoludique. En d'autres termes, il est nécessaire d'accepter qu'une partie de l'industrie vidéoludique est impossible à analyser, aussi bien sur le plan diachronique que synchronique. Cela rend donc caduque tous les potentiels débats sur les moteurs indisponibles pour le public.

Une deuxième conséquence, découlant de la première, est que le développement sur les consoles de jeux vidéo est nettement plus complexe à analyser et appréhender sur le plan technique. Cela notamment à cause des technologies utilisées qui se révèlent être, la plupart du temps, opaques. C'est pourquoi une étude comme la nôtre était difficile à mettre en place naturellement, c'est-à-dire sans impliquer le développement de notre approche sémio-informatique du DLT. Néanmoins, les difficultés substantielles rencontrées ont permis de créer de très nombreuses théories et structures, exploitables sur le plan scientifique. L'analyse des technologies opaques a donc été bénéfique pour perfectionner notre approche, mais nous estimons qu'elle n'est pas envisageable

sur un plan purement technique, contrairement aux technologies transparente et semi-transparente.

Une troisième conséquence est, pour notre corpus vidéoludique, que la totalité des jeux vidéo, que nous considérons comme étant des logiciels, sont des technologies intégralement opaques, en dehors des jeux vidéo *Open Source*. Comme pour les moteurs de jeu ou les API, cela signifie également que les éventuelles théories élaborées sur ces jeux vidéo sont exclusivement hypothétiques. Par exemple, une étude de *Grand Theft Auto III (*Rockstar Games 2001, PS2) pourrait seulement permettre de développer des hypothèses sur les aspects techniques du jeu, sans pour autant relever de faits purement objectifs. Néanmoins, cette théorie du degré d'agnosticisme développemental reste limitée pour appréhender pleinement la nature d'un moteur de jeu, y compris pour les moteurs opaques. C'est pourquoi notre prochaine sous-section va permettre l'élaboration de théories permettant de mieux esquisser la nature intrinsèque des technologies vidéoludiques.

4-2-C. FAISCEAUX ET SUBSTRATS VIRTUELS

Au regard des jeux en monde ouvert, nous constatons que la sixième génération représente une véritable évolution sur ce type d'univers virtuel, notamment grâce à des œuvres majeures comme *Grand Theft Auto III* (Rockstar Games 2001, PS2) et ses itérations successives. En effet, nous observons, depuis la publication de ce jeu vidéo, un nombre grandissant de jeux en monde ouvert, reprenant peu ou prou certaines caractéristiques de cette série influente et innovante (Barton 2019, 303). De surcroît, nous pouvons affirmer que les différents opus de la saga *Grand Theft Auto* (*ab*. 1997, PS1) sur la sixième génération sont réalisés avec *RenderWare*, mais cette technologie étant fermée, nous ne pouvons élaborer aucune théorie sur celle-ci. C'est pourquoi, plutôt que de réaliser simplement une analyse sur les signes interactifs, nous préférons élaborer des théories qui pourraient déterminer la nature intrinsèque des moteurs de jeu utilisés.

De surcroît, la démocratisation indéniable et massive des multiples moteurs de jeu lors de cette sixième génération permet de mettre en exergue des concepts éminemment pertinents pour notre analyse des univers virtuels. En particulier, nous pouvons développer une théorie inspirée de la philosophie, ayant pour objectif de mieux circonscrire la nature d'un univers virtuel mais également d'un moteur de jeu. Plus précisément, on va utiliser notre processus d'interthéoricité logicielle pour développer trois concepts majeurs que nous utiliserons dans notre approche du DLT :

- ► Le substrat virtuel, provenant de la théorie du substrat ;
- ► Le faisceau virtuel, inspiré de la théorie du faisceau ;
- ► La nature structurelle, pouvant être un hapax ou une instanciation.

Tout d'abord, c'est grâce à l'exportation originelle de l'interthéoricité logicielle que nous pouvons définir, avec Baptiste Le Bihan, la théorie du substrat comme étant « les objets matériels [qui] tirent leur identité à travers le temps et les mondes possible d'une substance sous-jacente qui porte les propriétés et que l'on peut appeler ''substrat'' » (Le Bihan 2016, 50). Désormais, il est possible d'effectuer une importation théorique de l'interthéoricité logicielle pour réutiliser ce concept sur les moteurs de jeu, ce que nous nommons substrat virtuel. On peut définir cela comme étant une distinction nette entre le moteur de jeu et

l'univers virtuel d'un jeu, ainsi que ses signes interactifs. Surtout, le substrat virtuel implique un univers virtuel du jeu ayant été conçu sans modifier le moteur, de quelque manière que ce soit. En d'autres termes, un moteur est un substrat distinct, qui ne doit subir aucune modification pour se conformer aux besoins d'un jeu vidéo.

À l'inverse, la théorie du faisceau expose l'idée, toujours selon Le Bihan, que « les objets matériels possèdent une identité temporelle et modale du fait de l'existence, non pas d'une substance sous-jacente, mais d'une relation qui attache les propriétés ensemble » (*ibid.*). De cette autre théorie philosophique qui s'oppose à la théorie du substrat peut découler, grâce à une importation théorique, ce que nous nommons le faisceau virtuel. Nous pouvons définir ce nouveau concept comme étant le développement d'un moteur de jeu *from scratch*, en même temps qu'un jeu et que ses interactions. L'objectif d'un tel moteur est de correspondre, autant que possible, aux spécificités d'un jeu vidéo.

Désormais, nous pouvons illustrer ces deux concepts à travers différents exemples du corpus vidéoludique de cette sixième génération de consoles. Ainsi, le moteur du jeu *Metal Gear Solid 3: Snake Eater* (Konami 2004, PS2) a été développé *from scratch*, et spécifiquement pour les besoins du jeu, comme mentionné dans la section précédente (Hivner 2003). Dans un tel cas, l'univers de *Metal Gear Solid 3: Snake Eater* (Konami 2004, PS2) est bien un faisceau virtuel, puisqu'il est *de facto* lié au moteur, et qu'il est potentiellement difficile de séparer le moteur de l'univers du jeu. La substance développementale des signes interactifs bas niveau est donc potentiellement liée au moteur de jeu. Par ailleurs, et au regard des informations mentionnées, il semblerait que le moteur de *Metal Gear Solid 3: Snake Eater* (Konami 2004, PS2) reprenne potentiellement une partie du moteur précédent, celui de *Metal Gear Solid 2: Sons of Liberty* (Konami 2001, PS2). Mais, parce qu'il s'agit de technologies cryptées, il n'est pas possible de savoir s'il existe des implications au regard de nos différentes théories. Nous ne pouvons donc pas vérifier si les modifications du moteur portent surtout sur l'interactivité ou uniquement sur une partie spécifique du moteur, comme le moteur de rendu 3D. Néanmoins, nos théories impliquent qu'un univers virtuel est un faisceau virtuel dès lors que le moteur de jeu est modifié pour répondre aux besoins de l'univers virtuel, même dans le cadre d'une modification infinitésimale.

En revanche, *Tom Clancy's Splinter Cell* (Ubisoft 2002, Multi), développé avec le moteur *Unreal Engine 2*, possède hypothétiquement un substrat virtuel. Ce même substrat conditionne la totalité des possibilités dans l'univers virtuel, et surtout les signes interactifs. En clair, les développeurs exploitent le moteur de jeu pour créer un univers virtuel, et utilisent donc cette substance pour produire le monde virtuel en question. Il existe également d'autres jeux vidéo que nous pouvons considérer comme étant des substrats virtuels, mais seulement de manière hypothétique. On pourrait illustrer cela avec le cas de *Grand Theft Auto III* (Rockstar Games 2001, PS2), pour lequel nous ne possédons pas d'informations suffisantes pour pouvoir définir s'il s'agit d'un faisceau virtuel ou d'un substrat virtuel. Et, dans une telle situation, plutôt que d'envisager le substrat virtuel, nous considérons simplement qu'il se trouve dans une situation indéterminée.

En réalité, l'intérêt de ce concept est de différencier deux approches qui peuvent altérer le développement d'un jeu vidéo, et surtout son interactivité ou son *gameplay*. Ainsi, la création d'un moteur de jeu *ex nihilo*, et dont les caractéristiques répondent spécifiquement aux besoins d'un jeu précis, permet de considérer que sa nature intrinsèque est intégralement liée au jeu en question. Surtout, ce faisceau virtuel nuance la définition d'un moteur de jeu qui impliquait une nette séparation entre les *assets* du jeu et le moteur en lui-même. Notre objectif est de considérer qu'un moteur de jeu spécialement conçu pour un jeu précis, même s'il n'intègre pas des *assets*, n'en reste pas moins totalement lié à l'univers du jeu. Ce même moteur ne pourrait aucunement être indépendant du jeu en question, et pourrait posséder des fonctionnalités exclusivement pour un jeu. Or, le substrat virtuel implique plutôt le développement d'un jeu vidéo de manière totalement indépendante du moteur. Ce même jeu doit hypothétiquement adapter les caractéristiques de l'univers virtuel au moteur de jeu.

Cette double théorisation reflète ainsi une préoccupation dans le développement d'un jeu, en particulier de l'adaptation d'un jeu au moteur ou son contraire. En effet, et pour reprendre les propos de Wang et Nordmark, il s'agit surtout d'un « compromis entre la liberté créative et les limitations techniques, [où] soit les idées doivent s'adapter à la technologie, soit la technologie aux

idées » (Wang et Nordmark 2015, 276 TDR)[479]. C'est dans ce cas où le faisceau virtuel adapte directement la technologie à l'idée. Or, le substrat virtuel va plutôt conformer l'idée d'un jeu à une technologie particulière. Certes, dans les deux cas, cela implique de nombreuses limitations à prendre en compte, mais le substrat virtuel possède des limitations beaucoup plus importantes, où le moteur impose une hypothétique vision d'un jeu. En revanche, les limitations induites par le faisceau virtuel sont surtout matérielles, voire budgétaires.

Au-delà de ces deux notions bien distinctes, il nous semble important d'introduire un troisième concept inédit, que nous nommons nature structurelle. L'objectif de cette théorie est de mieux circonscrire, voire de nuancer la nature d'un univers virtuel, et de mettre cela en corrélation à la fois avec le faisceau virtuel et le substrat virtuel. Ce même concept va être subdivisé en deux catégories exclusives et inédites pour définir un univers virtuel : un hapax structurel ou une instance structurelle.

Désormais, nous pouvons développer le premier sous-concept du hapax structurel, que nous définissons comme étant un univers virtuel possédant des caractéristiques si uniques que cela implique hypothétiquement la conception d'un moteur de jeu *from scratch*. Par exemple, le jeu *Animal Crossing: Population Growing!* (Nintendo 2001, GameCube) semble posséder un hapax structurel, et le développeur du jeu, dans une entrevue réalisée par *Nintendo UK*, explique justement que « nous n'avions pas de genre adéquat que nous pouvions utiliser comme référence, c'était très difficile d'expliquer (...) comment le promouvoir » (Nintendo UK 2016 TDR)[480].

Surtout, selon le magazine de jeux vidéo *Edge*, ce jeu *GameCube* « était initialement publié comme étant un titre N64 exclusivement au Japon en 2001 (...) [et] avait été commandé simplement comme moyen d'utiliser l'horloge en temps réel intégrée du projet infructueux de la 64DD de Nintendo » (Edge 2008 TDR)[481]. Le fait de développer un jeu avec des besoins uniques, comme ce cas

479. « trade-off between creative freedom and the technical limitations, [where] either the ideas must be adapted to the technology, or the technology to the ideas » (Wang et Nordmark 2015, 276).

480. « we didn't have a proper genre that we could use as a reference point, it was very difficult to explain (...) how to promote it » (Nintendo.co.uk 2016).

481. « was first published as a Japan-only N64 title in 2001 (...) [and] was commissioned merely as a way to utilise the realtime clock built into Nintendo's failed 64DD » (Edge 2008).

de figure, pourrait nécessiter une modification d'un hypothétique moteur de jeu. Précisons toutefois que nous n'avons pas d'informations sur le développement de ce jeu en particulier.

En revanche, nous définissons notre concept inédit nommé instance structurelle comme étant un univers virtuel dont les fonctionnalités sont reproductibles sur un moteur de jeu, sans modification aucune. Ce cas permet surtout de mettre en exergue les jeux élaborés avec des moteurs conçus *from scratch* sans qu'il existe une justification exclusivement technique. Naturellement, de nombreux facteurs peuvent justifier la création d'un moteur de jeu spécifique pour un jeu ou une série de jeux. Dans notre cas, cela permet de distinguer simplement le faisceau virtuel, impliquant un jeu entièrement lié à son moteur, d'un même moteur de jeu modifié ou conçu spécifiquement pour un jeu.

Désormais, nous pouvons synthétiser ces multiples concepts dans le tableau ci-dessous, c'est-à-dire aussi bien le substrat que le faisceau virtuel, tout en incluant les deux variations de la nature structurelle.

Notion	Définition
Substrat virtuel	Utilisation d'un moteur de jeu indépendant d'un univers virtuel, sans altération pour correspondre aux besoins d'un jeu précis, qui s'adapte aux contraintes du moteur
Faisceau virtuel	Développement d'un moteur de jeu *from scratch* dans le but de créer un jeu vidéo, ayant des signes interactifs uniques ou génériques
Hapax structurel	Nature structurelle d'un univers virtuel possédant des caractéristiques uniques et impliquant la création d'un moteur de jeu *from scratch*
Instanciation structurelle	Nature structurelle d'un univers virtuel possédant des fonctionnalités reproductibles sur un moteur de jeu quelconque

Tableau 30. Faisceaux et substrats virtuels

En substance, nos définitions de ces concepts majeurs, le substrat virtuel, le faisceau virtuel et la nature structurelle, mettent en exergue une interrogation fondamentale sur l'unicité des signes interactifs d'un jeu. En effet, on pourrait se demander dans quelle mesure le développement d'un jeu vidéo avec un

gameplay reproductible nécessiterait la création d'un moteur *from scratch*, et pourquoi des limitations strictement technologiques persisteraient dans les interactions. Et, dans les chapitres ultérieurs, nous réutiliserons ces multiples théories pour mettre en exergue plusieurs pistes au regard de ces mêmes limitations.

4-3. SYNTHÈSE DE LA SIXIÈME GÉNÉRATION

Notre quatrième chapitre était centré sur la sixième génération de console, ayant débuté dès 1998 avec la *Dreamcast*, la *PlayStation 2*, la *Xbox* et la *GameCube*. Nous avons principalement traité de la première démocratisation des moteurs de jeu AAA ainsi que des outils de développement indépendants, puis nous avons également élaboré des théories sémio-informatiques qui ont renforcé celles du chapitre précédent.

4-1. TECHNOLOGISATION VIDÉOLUDIQUE

La première section majeure de ce chapitre a permis de mettre en relief l'incorporation du *C++* puis la commercialisation de moteurs de jeu AAA majeurs. En outre, nous avons mis en exergue des outils de développement indépendants, notamment le *YA-Basic* et le *Linux Kit* de la *PlayStation 2*, ainsi que des moteurs propriétaires ou *Open Source*.

L'introduction de cette première section a contextualisé l'apparition des consoles majeures lors de la sixième génération de consoles, débutant par la *Dreamcast* en 1998 (DeMaria 2019b ; Wardyga 2019). Néanmoins, la commercialisation de cette dernière cessera dès 2001 (Williams 2017), malgré plus de 600 œuvres (Wardyga 2019). Sony a également commercialisé la *PlayStation 2*, avec pour objectif de la transformer en centre de divertissement domestique (Mainelli 2000 ; Xbox 2021a). Plus encore, le but principal de *Sony* était de pouvoir remplacer les ordinateurs par des *PlayStation 2* (CNET Australia 2008). Cette situation expliquerait, en partie, pourquoi *Microsoft* a élaboré la console Xbox (Xbox 2021a).

La première sous-section nommée **Incorporation du C++** a d'abord évoqué les hypothétiques difficultés substantielles du développement de jeux sur *PlayStation 2* (Dutton 2012 ; Kent 2000 ; Kent 2001 ; White 2002). Nous avons ensuite présenté le langage *C++*, élaboré par Stroustrup en 1985 (Liang *et al.* 2014 ; Petzold 2023). Ce langage reste encore le plus utilisé dans l'industrie vidéoludique (Gregory 2019), mais également l'un des plus populaires encore en 2023 (TIOBE 2023 ; O'Grady 2023 ; Cass 2023). Ainsi, le *C++* a probablement été massivement utilisé pour le développement de jeux de la sixième génération, mais nous n'avons pas de sources scientifiques pouvant attester d'une telle

situation. Toutefois, cela a permis de mettre en avant les paradigmes de programmation, et particulièrement la programmation orientée objet ayant émergée dans les années 1960 (Clark 2013) puis s'étant démocratisée dans les années 1980 (Sebesta 2016). Sur ce point, nous avons évoqué les quatre piliers de la POO, notamment l'encapsulation, l'abstraction, l'héritage et le polymorphisme (Horton et Van Weert 2020).

Puis la deuxième sous-section, nommée **Développement des moteurs de jeu AAA**, a mis en exergue le développement des moteurs AAA, notamment les deux versions de *RenderWare*, *Platform* et *Studio* (Pedersen 2003), ainsi que le moteur *Unreal Engine 2* (Pranckevičius 2018). Le nombre de moteurs semblait si significatif que Pedersen (2003) a classifié ces derniers en trois catégories distinctes : moteurs économiques, moteurs moyens et moteurs luxueux. Nous avons également repris la notion de moteur *indoor*, spécialisé pour les rendus d'environnement en intérieur, et moteur *outdoor* pour les moteurs optimisés pour les lieux virtuels en extérieur (Gregory 2019 ; Hivner 2003). Puis nous avons différencié un moteur de jeu, l'outil complet permettant de créer des jeux vidéo, d'un moteur de rendu, un outil gérant exclusivement l'affichage graphique d'une application 3D. On a également constaté que chaque moteur est développé d'une manière unique, et la délimitation entre un moteur et son jeu peut varier considérablement selon les moteurs (Gregory 2019 ; McShaffry et Graham 2013).

Enfin, la troisième sous-section nommée **Démocratisation des outils indépendants** a permis de mettre en relief des possibilités du développement amateur, d'abord sur la console *PlayStation 2* via le *Linux Kit* ainsi que le *YA-Basic*. Ainsi, le *Linux Kit* permet la conception de prototypes, voire de moteurs de jeu en 3D tout en ayant des limitations substantielles par rapport au développement professionnel (Fackroth *et al*. 2005). Quant au langage *YA-Basic*, basé sur le *BASIC* et uniquement disponible sur les consoles européennes, celui-ci semblait être une démonstration technique hypothétiquement liée à une taxe d'importation (Smith 2000). Ensuite, nous avons distingué entre deux types de moteurs : les moteurs de jeu *Open Source* et les moteurs propriétaires prévus pour les indépendants. La première sous-catégorie nous a permis de mentionner trois moteurs, *Ogre, Crystal Space* et *Irrlicht* (Peternier, Vexo et Thalmann 2008), mais également d'illustrer concrètement la distinction entre moteurs de jeu et

moteurs graphiques. Nous avons également fait un comparatif avec l'outil *DarkBasic Pro*, lequel possédant des limitations de performance et d'optimisation (Harbour et Smith 2003), contrairement aux moteurs commerciaux comme *Unreal Engine 2*.

4-2. CARACTÈRES PRIMORDIAUX INTERACTIFS

Dans la deuxième section de ce chapitre, nous avons complexifié nos théories sémio-informatiques, notamment avec les caractères primordiaux de l'interactivité, l'aspect générationnel, le manichéisme des interactants et enfin les faisceaux et substrats des univers virtuels.

En **introduction**, nous avons précisé la complexité de notre signe interactif en incorporant trois caractères primordiaux interactifs : l'arbitraire, la linéarité apparente et la complexité des instructions. Les deux premiers concepts proviennent du CLG de Saussure, le troisième s'appuie en revanche sur la pénalité d'abstraction en informatique (Williams 2019). L'arbitraire du signe implique de pouvoir remplacer un signifiant par un autre, tout en conservant un signifié identique. Un phénomène illustré par le signe interactif symbolisant le fait d'entrer à l'intérieur d'un véhicule dans *Grand Theft Auto III* (Rockstar Games 2001, PS2). L*'input,* ou le signifiant, permettant de déclencher une réaction varie selon la version *PlayStation 2*, *Xbox* ou *PC* du jeu, tout en conservant le même signifié. Puis la linéarité apparente a indiqué l'action puis la réaction d'un seul et unique signe interactif, tout en considérant que la linéarité est apparente au regard de certains concepts de programmation comme le *multithreading*, mais aussi à la nature intrinsèquement non-linéaire d'un univers virtuel. Ce deuxième caractère primordial permet surtout d'isoler un signe interactif. Nous avons également défini le troisième caractère, celui de complexité des instructions, comme étant le nombre d'instructions nécessaires pour réaliser un signe interactif comme l'ouverture d'une porte ou la translation d'un objet.

Puis nous avons développé dans la première sous-section, **Interactivité générationnelle**, trois autres concepts importants pour notre signe interactif : l'immutabilité interactive, la mutabilité interactive et l'ancrage générationnel. L'objectif principal de ces concepts était de vérifier si l'interactant peut agir sur les signes interactifs et les altérer, et si ces signes sont présents à travers plusieurs jeux vidéo, voire plusieurs générations de consoles. Ainsi, l'immutabilité

interactive représente des signes interactifs qui n'ont aucune altération possible. Or, la mutabilité interactive est la possibilité pour un interactant d'altérer un signe interactif. Nous avons ensuite subdivisé l'ancrage générationnel en deux : un ancrage intragénérationnel où un signe interactif est présent dans plusieurs jeux différents d'une même génération. Puis nous avons également conceptualisé l'ancrage intergénérationnel pour des signes présents dans plusieurs générations différentes.

Ensuite, à partir de la deuxième sous-section intitulée **Degré d'agnosticisme développemental**, nous avons élaboré un concept majeur pour permettre de mettre en exergue les différentes implications des technologies disponibles. Il s'agit des trois degrés d'agnosticisme développemental, s'appuyant sur le concept informatique d'agnosticisme des technologies (Wright 2022). Ainsi, le degré d'agnosticisme zéro, que nous avons également nommé technologies transparentes, implique qu'une technologie est disponible pour le grand public, y compris son code source. C'est notamment le cas pour les moteurs graphiques comme *Irrlicht* ou *OGRE*. En utilisant notre outil de la comparaison anachronique, on constate également que le moteur *Unreal Engine 4* est une technologie transparente. Puis le degré d'agnosticisme un, aussi nommé technologies semi-transparentes, implique l'existence de moteurs disponibles pour le grand public, mais dont le code source est indisponible. Toujours en reprenant la comparaison anachronique, nous avons illustré cela via le moteur *Unity*. Enfin, le degré d'agnosticisme deux, que nous avons intitulé technologies opaques, a été important pour mettre en lumière les technologies indisponibles pour le grand public, par exemple *RenderWare*. Les trois conséquences de cet agnosticisme développemental sont importantes pour une étude scientifique : une difficulté dans la réalisation d'une étude sur des technologies opaques ; la complexité substantielle à analyser le développement de jeux sur consoles, utilisant des technologies opaques ; enfin, l'analyse des jeux vidéo qui relève également de technologies intrinsèquement opaques.

Enfin, la troisième sous-section **Faisceaux et substrats virtuels** nous a permis d'intégrer deux théories philosophiques en utilisant l'interthéoricité logicielle. Nous avons d'abord érigé le concept du substrat virtuel, provenant de la théorie philosophique du substrat (Le Bihan 2016). Celui-ci implique une distinction nette entre le moteur de jeu, d'une part, et l'univers virtuel ainsi que les signes

interactifs, d'autre part. Le moteur de jeu est donc un substrat qui ne va subir aucune modification pour répondre à des besoins particuliers pour la création d'un jeu vidéo. *Tom Clancy's Splinter Cell* (Ubisoft 2002, Multi) est hypothétiquement l'exemple d'un substrat virtuel, où le jeu a été développé sans modification du moteur *Unreal Engine 2*, d'après nos recherches. Puis nous avons développé le concept du faisceau virtuel, reprenant la théorie philosophique du faisceau (Le Bihan 2016). Ce deuxième concept implique la création d'un moteur de jeu *from scratch*, en même temps qu'un jeu vidéo et que ses mécanismes interactifs. Cela signifie *de facto* qu'un tel moteur est élaboré spécifiquement pour répondre aux besoins d'un jeu vidéo, contrairement au substrat qui en est indépendant. Nous avons illustré le concept du faisceau par l'exemple de *Metal Gear Solid 3: Snake Eater* (Konami 2004, PS2), dont le moteur a été réalisé pour répondre aux besoins du jeu (Hivner 2003). Cette double théorisation du faisceau et substrat virtuel est liée à une préoccupation majeure dans le développement de jeux vidéo, celle de l'adaptation d'un projet de jeu à une technologie ou son contraire (Wang et Nordmark 2015). Au-delà de ces deux notions, nous avons enfin développé un troisième concept inédit, la nature structurelle, permettant de mieux nuancer la nature d'un univers virtuel au regard du faisceau virtuel ainsi que du substrat virtuel. Ce concept est subdivisé en deux : hapax structurel et instance structurelle. Nous avons défini un hapax structurel comme étant un univers virtuel possédant des caractéristiques si uniques qu'il est nécessaire de créer un moteur de jeu *from scratch* pour l'adapter, c'est-à-dire un faisceau virtuel. Ce concept a été illustré par *Animal Crossing: Population Growing!* (Nintendo 2001, GameCube). Le dernier concept de l'instance structurelle signifie qu'un univers virtuel possède des fonctionnalités qu'il est possible d'implémenter dans un moteur de jeu, sans modifier ce dernier.

5. SEPTIÈME GÉNÉRATION : POLYMORPHIE DES MOTEURS 3D

5-1. SUPER-ORDINATEURS DE DIVERTISSEMENT

L'industrie vidéoludique a évolué substantiellement à partir de la septième génération de consoles, plus précisément dès 2005. Une telle situation rendue possible grâce à une puissance technologique nouvelle ainsi qu'à une démocratisation importante des outils de développement. Surtout, cette génération est portée par trois consoles majeures : la *Xbox 360*, la *PlayStation 3* ainsi que la *Wii*. Ainsi, la première console à être distribuée est la *Xbox 360*, dès novembre 2005 aux États-Unis (Wardyga 2019, 330). Cette dernière a vraisemblablement joué un rôle important selon Hicks, propulsant le jeu « Gears of War [qui] a fait des jeux de tir (...) le genre phare de l'époque (...) [et] a établi Unreal Engine comme étant la principale plate-forme de développement next-gen » (Hicks 2013 TDR)[482]. Un genre éminemment lié à la genèse des moteurs de jeu en 3D temps réel, mais également à la *Xbox* elle-même, notamment avec la série *Halo* (*ab*. 2001, Xbox), comme observé dans le chapitre précédent de notre thèse (Barton 2019).

Par la suite, nous pouvons constater que la console *PlayStation 3*, distribuée en novembre 2006 au Japon et aux États-Unis (Wardyga 2019, 304), est présentée par Kutaragi « comme une machine de divertissement et de media polyvalente (...), un "super-ordinateur de divertissement" » (CNET Australia 2008 TDR)[483], autrement dit un *media center*. Cette stratégie semble être similaire à celle adoptée pour la *PlayStation 2*. Un même objectif qui se traduisait par de nombreuses caractéristiques ajoutées sur la console, notamment la possibilité d'installer le système d'exploitation *Linux*. Une possibilité vraisemblablement retirée par la suite pour des raisons de sécurité. Enfin, la distribution de la *Wii* s'est déroulée, comme le rappelle Wardyga, « aux États-Unis avant le Japon, (...) au Times Square de New York le 19 novembre 2006 » (Wardyga 2019, 312 TDR)[484].

482. « Gears of War made (...) shooters the signature genre of the era (...) [and] established Unreal Engine as the de facto next-gen development platform » (Hicks 2013).
483 . « as an all-around entertainment and media machine (...), an "entertainment supercomputer" » (CNET Australia 2008).
484. « in the United States before Japan, (...) in New York's Times Square on November 19, 2006 » (Wardyga 2019, 312).

Par ailleurs, un autre événement va restructurer l'industrie vidéoludique entière : l'usage important d'internet sur les consoles. Les conséquences sont éminemment nombreuses, allant bien au-delà de notre problématique, mais nous pouvons mettre en exergue deux aspects principaux : l'émergence des boutiques en ligne ainsi que les mises à jour logicielles pour les jeux vidéo commercialisés. En effet, on assiste également à la démocratisation des boutiques virtuelles pour les trois consoles, selon Williams, où « Microsoft a relancé le Xbox Live Arcade (...) en 2005, suivi de près par le PlayStation Network Store de Sony en 2006, et le Wii Shop Channel de Nintendo en 2008 » (Williams 2017, 207 TDR)[485]. Sur PC, la boutique en ligne *Steam*, déjà lancée « en 2003 (...), a commencé à vendre des jeux des développeurs tiers en 2005 » (*ibid*. TDR)[486]. Et la plate-forme *Steam* jouera un rôle essentiel pour une diversification du marché vidéoludique, permettant aux développeurs AAA, indépendants et amateurs de publier leurs jeux vidéo.

Et, au regard des mises à jour logicielles, on constate que cette possibilité, désormais démocratisée, va restructurer l'industrie vidéoludique. Ainsi, jusqu'à la sixième génération incluse, les jeux vidéo étaient généralement développés et publiés sans pouvoir effectuer des mises à jour importantes, sauf à travers des disques d'expansion. Certes, le *Xbox Live* était fonctionnel pour la sixième génération, mais celui-ci a désormais substantiellement évolué. De plus, dès la septième génération, il devenait possible d'ajouter du contenu additionnel ou de publier des mises à jour pour un jeu vidéo, ouvrant le marché à des possibilités nouvelles. Par exemple, c'est grâce à cette caractéristique qu'on assistait à une nouvelle forme de disponibilité d'un jeu, nommée Accès Anticipé ou *Early Access* en anglais, phénomène ayant, selon Austen Goslin, « commencé en 2009 avec Minecraft (...) [et] a donné aux développeurs un moyen de gagner de l'argent tout en finissant le jeu » (Goslin 2019 TDR)[487]. Autrement dit, nous pouvons

485. « Microsoft relaunched Xbox Live Arcade (...) in 2005, closely followed by Sony's PlayStation Network Store in 2006, and Nintendo's 2008 Wii Shop Channel » (Williams 2017, 207).
486. « in 2003 (...) began selling games from third-party developers in 2005 » (Williams 2017, 207).
487. « started in 2009 with Minecraft (...) [and] gave developers a way to make money while still finishing the game » (Goslin 2019).

considérer que les jeux deviennent progressivement évolutifs et adaptables, en particulier par rapport au retour des joueurs.

Nous pouvons désormais présenter les différentes consoles de cette génération dans le tableau ci-dessous, en reprenant les informations avancées par Wardyga (2019, 318).

Consoles	Année de distribution	Ventes mondiales approximatives
Sony PlayStation 3	2006	Supérieur à 85 millions
Nintendo Wii	2006	Supérieur à 100 millions
Microsoft Xbox 360	2005	Supérieur à 85 millions

Tableau 31. Ventes mondiales de la septième génération (Wardyga 2019, 318)

Ainsi, au regard de la situation de cette septième génération, nous estimons qu'il est important de se concentrer sur plusieurs aspects tout au long des trois sous-sections successives, aussi bien concernant différents moteurs de jeu propriétaires que deux moteurs de jeu qui étaient disponibles pour le grand public : *Unreal Development Kit* et *Unity*. L'objectif étant de vérifier en quoi ceux-ci pouvaient jouer un rôle potentiel dans d'éventuelles limitations sur l'interactivité.

5-1-A. Généralisation des moteurs 3D internes

Au regard de l'architecture des multiples consoles de la septième génération, il semble que celle de la *PlayStation 3* soit considérée comme étant la plus complexe pour le développement de jeux vidéo. Cette même complexité semble provenir de son architecture relativement unique, un fait posant des obstacles éminemment importants dans le cadre de la création de jeux. Cela principalement pour deux raisons : une difficulté d'apprentissage supposée de cette nouvelle architecture mais également pour le développement multiplateforme. Pourtant, au départ, il était plutôt question d'une supposé facilité de développement de jeux sur *PlayStation 3*, puisque Kent rappelle que « Sweeney a présenté la spectaculaire démo technique Unreal Engine (...) lors de sa présentation à l'E3 2005 (...) [et] l'a construit en deux mois à peine » (Kent 2021, ch. 13 TDR)[488]. Mais, par la suite, des développeurs ayant tenté d'utiliser la machine ont expliqué qu'un « logiciel qui exploite le potentiel de la Cell requiert un effort de développement nettement plus important que sur des plates-formes traditionnelles » (Scarpazza, Villa et Petrini 2007 TDR)[489].

C'est pourquoi on peut insister en réitérant nos propos, grâce à Kent qui indique que la « PlayStation 3 avait la réputation d'être particulièrement difficile à programmer » (Kent 2021, ch. 13 TDR)[490]. Une difficulté qui dénote un potentiel retard dans le développement des jeux, également accentuée par le fait que, selon Raymond Graham, les « SPU (...) fonctionnent vraiment différemment de toutes les autres plates-formes disponibles, et donc être capable de faire (...) du multi-plateforme (…) est certainement un défi » (Graham 2018, 100 TDR)[491]. De surcroît, au regard de la démonstration technique du moteur *Unreal Engine 3* présentée en 2005 et dont le portage semblait relativement peu complexe pour Sweeney, Kent considère qu'il s'agit d'une situation peu commune puisque, si

488. « Sweeney showed the spectacular Unreal Engine tech demo (...) during his E3 2005 presentation (...) [and] built it in two short months » (Kent 2021, ch. 13).
489. « software that exploits the Cell's potential requires a development effort significantly greater than traditional platforms » (Scarpazza, Villa et Petrini 2007).
490. « PlayStation 3 had a reputation for being particularly difficult to program » (Kent 2021, ch. 13).
491. « SPU (...) works really differently from every other platform around, and so being able to do (...) cross-platform (…) is definitely a challenge » (Graham 2018, 100).

« Sweeney avait réussi à faire un portage de sa démo en deux mois, (...) il était largement considéré comme l'un des grands intellectuels de l'industrie » (Kent 2021, ch. 13 TDR)[492].

Néanmoins, on observe bien que la programmation console conservait encore les deux langages majoritairement cités dans les générations précédentes, selon Graham, puisque « les langages de programmation (...) de la PS3 et des SPU (…) étaient soit en C, soit en C++ » (Graham 2018, 101 TDR)[493]. Mais, au regard de la situation pour la génération précédente, si les « graphismes et la puissance de traitement n'avaient jamais été un problème avec la marque PlayStation, la complexité de programmation des jeux, en revanche, si » (Kent 2021, ch. 12 TDR)[494]. On pourrait surtout s'interroger sur la complexité de cette architecture propriétaire de la *PlayStation 3*, car elle est vraisemblablement distincte des ordinateurs de bureau, en plus d'introduire des CPU multi-cœurs. En effet, Gregory insiste sur ce dernier aspect, où il indique que la *PlayStation 3* « contient un seul CPU principal appelé Power Processing Unit (PPU), huit coprocesseurs nommés Synergistic Processing Units (SPUs), et un NVIDIA RSX Graphics Processing Unit (GPU) » (Gregory 2019, 199 TDR)[495]. Naturellement, les détails techniques de l'architecture, voire de la programmation sur des architectures multi-cœurs sont hors du cadre de notre recherche. Néanmoins, les composants de la console possèdent des conséquences importantes pour la création de jeu.

Cela étant, un point nettement plus pertinent qu'il est possible de mettre en exergue, pour cette septième génération, est celui de la standardisation et la structuration majeure de l'industrie vidéoludique. Cela est notamment observable à travers de nombreux moteurs de jeu propriétaires développés par différents studios, même si les informations sont peu accessibles au regard de la nature secrète de l'industrie vidéoludique, comme indiqué dans notre recherche. En effet, nous souhaitions évoquer des moteurs propriétaires et

492. « Sweeney had been able to port his demo over in two months, (...) he was widely considered one of the great intellects of the industry » (Kent 2021, ch. 13).
493. « PS3 and SPU (…) programming language is either C or C++ » (Graham 2018, 101).
494. les « graphics and processing power had never been a problem with the PlayStation brand, the complexity of programming games (...) was » (Kent 2021, ch. 12).
495. « contains a single main CPU known as the Power Processing Unit (PPU), eight coprocessors known as Synergistic Processing Units (SPUs), and an NVIDIA RSX Graphics Processing Unit (GPU) » (Gregory 2019, 199).

développés en interne comme le *Crystal Tools*, anciennement nommé *White Engine et* utilisé par l'entreprise *Square Enix* pour des jeux comme *Final Fantasy XIII* (Square Enix 2009, Multi). Il existe également le moteur *RAGE*, utilisé pour développer les différents opus de *Grand Theft Auto* (ab. 1997, PS1) à partir de la quatrième itération, et relativement pertinent pour notre problématique. Cependant, il n'était pas possible de s'appuyer sur des sources scientifiques pour étayer notre propos, c'est pourquoi nous avons éludé ces mêmes moteurs internes.

Néanmoins, on pourrait également mettre en exergue des moteurs commercialisés et disponibles pour d'autres développeurs, comme *GameBryo*. Nous constatons, pour ce dernier moteur et grâce à Steve Klett, que, « une fois que les développeurs apprennent à utiliser le moteur Gamebryo, ils peuvent l'utiliser pour n'importe quel type de jeu, quel que soit le genre ou le sujet » (Klett 2004, 58 TDR)[496]. Ce qui impliquait donc une forme de liberté de création importante au regard des caractéristiques du *gameplay*. Cependant, ces mêmes moteurs sont commercialisés exclusivement pour des développeurs professionnels, rendant impossible une quelconque analyse de la démocratisation des outils de développement de jeux dans la sphère publique, du moins pour ces outils en question. Or, notre objectif est plutôt de mettre en lumière la publication de plusieurs moteurs qui sont à la fois utilisables pour des professionnels, mais également pour le grand public. C'est pourquoi nous pouvons désormais consacrer notre prochaine sous-section à ce sujet.

496. « once developers learn how to use the Gamebryo engine, they can use it for any type of game, regardless of genre or subject matter » (Klett 2004, 58).

5-1-B. UNIFICATION INDÉPENDANTE

Au-delà d'une première forme de standardisation des moteurs propriétaires triple A, la septième génération de consoles symbolise surtout une démocratisation des outils de création de jeux, mais également une convergence entre outils dit professionnels et amateurs. D'abord, au-delà des outils simplistes, c'est surtout l'outil *XNA* de Microsoft qui débuta cette popularisation, selon Sherrod, puisque « XNA est devenu un excellent moyen pour les amateurs, étudiants et développeurs indépendants de créer (...) des jeux » (Sherrod 2008, 12 TDR)[497]. Un outil permettant de développer des jeux vidéo aussi bien sur PC que sur la console *Xbox 360*. Sherrod souligne ensuite, probablement en référence à des consoles de développement amateur que nous avons déjà évoquées, notamment la *Net Yaroze* ou encore le *Linux Kit*, qu'il ne s'agit pas de la première tentative. Néanmoins, on peut considérer, toujours grâce à Sherrod, que les employés de Microsoft « sont les premiers à le faire d'une manière qui a véritablement aidé » (*ibid.* TDR) [498] , autrement dit qui a favorisé une démocratisation importante du développement de jeux vidéo.

De surcroît, nous pouvons également observer, grâce à Bossom et Dunning, que la « nouvelle vague de développement de jeux indépendants est en partie due aux ordinateurs à faible coût, à la facilité d'accès aux logiciels puissants (...), et à l'émergence d'outils gratuits ou *open source* » (Bossom et Dunning 2015, 31 TDR)[499]. Ainsi, nous pouvons mettre en exergue la démocratisation de plusieurs outils principaux : *CryEngine*, *Unity* ou encore *Unreal Development Kit*, la version gratuite et grand public du moteur *Unreal Engine 3*. Quant à la *PlayStation 3*, nous pouvons surtout évoquer *PhyreEngine*, un moteur qui, d'après Martin Middleton, « était disponible pour tout développeur PlayStation ; on pouvait simplement le télécharger depuis leur forum de développement » (Middleton 2018, 82 TDR)[500].

497. « XNA has become a great way for hobbyists, students, and independent game developers to create (...) games » (Sherrod 2008, 12).
498. « are the first to do it in a way that has really helped » (Sherrod 2008, 12).
499. « new wave of indie game development is partly a consequence of low-cost computing, ease of access to powerful (...) software, and the emergence of freeware or open source software tools » (Bossom et Dunning 2015, 31).
500. « was available for anyone who was a PlayStation developer; you could just download it from their dev forum » (Middleton 2018, 82).

C'était la première fois que Sony mettait à disposition un moteur de jeu complet et gratuit, pour les développeurs officiels. Il convient de noter que le *PhyreEngine* se démarquait de manière significative, puisque « Sony a également rendu le code source de Phyre disponible (...), alors que Unity est bien plus semblable à une boîte noire » (*ibid*. TDR)[501], puisque le code source du moteur *Unity* n'était pas disponible. Néanmoins, comme indiqué, le *PhyreEngine* était malgré cela un moteur disponible uniquement pour les développeurs *PlayStation*, et les informations sur celui-ci sont très parcellaires.

Les multiples aspects du moteur *UDK* seront abordés plus loin. Néanmoins, nous estimons qu'il pourrait être judicieux de comparer la situation de l'époque du moteur *UDK* avec celle de *Unity*. Et, au regard du moteur *Unity*, nous pouvons constater une distribution peu avant le début de cette génération, comme le rappelle Oli Wilkinson, puisque « la première version publique du moteur Unity a été publiée en juin 2005, peu de temps après la sortie de Gooball, un jeu développé avec une version *pre-release* du moteur » (Wilkinson 2009 TDR)[502]. En revanche, c'est seulement en mars 2009, d'après la publication de Tom Higgins, que l'entreprise Unity « a distribué Unity 2.5 (...) offrant la prise en charge de Unity sur Mac OS X et Windows » (Higgins 2009 TDR)[503], autrement dit la même année que la version publique du moteur *Unreal Engine 3*, nommée *Unreal Development Kit* (ou *UDK*).

Ainsi, la démocratisation des moteurs de jeu professionnels pour le grand public, c'est-à-dire accessible sans licence professionnelle, a été un événement capital dans l'industrie vidéoludique. C'est pourquoi on peut constater, avec Benjamin Nicoll et Brendan Keogh, que ces multiples « changements radicaux dans la disponibilité et l'accessibilité de divers moteurs de jeu (...) ont créé un terreau fertile pour une pluralité de styles de jeux vidéo [et] de genres » (Nicoll et Keogh 2019, 3 TDR)[504]. Surtout, Matthew Burns, un ancien développeur chez

501. « Sony also made Phyre's source code available (...), whereas Unity is much more of a black box » (Middleton 2018, 82).
502. « the first public version of the Unity engine was released in June 2005, shortly after the release of Gooball - a game developed with a pre-release version of the engine » (Wilkinson 2009).
503. « shipped Unity 2.5 (...) offers support for Unity authoring on Mac OS X and Windows » (Higgins 2009).
504 « radical shifts in the availability and accessibility of various game engine (...) has created fertile ground for a plurality of videogame styles, [and] genres » (Nicoll et Keogh 2019, 3)

Bungie, va mettre en exergue le fait que « Unity Pro 3 offre aux développeurs de jeux de tous niveaux d'expérience une plate-forme flexible (...) pour concrétiser leurs idées » (Burns 2010, 36 TDR)[505]. C'est précisément la raison expliquant pourquoi nous pouvons évoquer une première convergence des indépendants et studios AAA vers les mêmes outils de développement, notamment *Unity*, *PhyreEngine* ou encore *Unreal Development Kit*.

Et, par la suite, on constate avec Jon Brodkin que des moteurs comme « Unity3D et les produits similaires (...) ont contribué à démocratiser les outils de développement de jeux (...) utilisés par (...) des entreprises de jeux à un coût minime ou nul » (Brodkin 2013 TDR)[506]. En effet, nous avons constaté une démocratisation relativement importante du développement de jeux, en utilisant un autre modèle économique que, par exemple, les moteurs de jeu cités par Pedersen (2003) dans la génération précédente. C'est pourquoi nous ne pourrions pas reprendre la classification des moteurs de Pedersen, notamment des moteurs économiques, moyens et luxueux, pour savoir dans quelle catégorie le moteur *Unity* peut être placé. Celui-ci étant, en effet, utilisable aussi bien pour des professionnels que des indépendants ou des amateurs. En revanche, nous pouvons constater que la disponibilité publique de ce même moteur, par une entreprise majeure de surcroît, est une nouveauté qui n'existait pas dans les exemples des générations antérieures. Et il en est également de même pour le moteur UDK, qui fait l'objet de notre prochaine section.

505. « Unity Pro 3 offers game developers with all levels of experience a flexible (...) platform in which to realize ideas » (Burns 2010, 36).

506. « Unity3D and similar products (...) helped democratize game development (...) tools used by (...) game companies available to developers at little or no cost » (Brodkin 2013).

5-1-C. Kit de développement protéiforme

Le développement de jeux vidéo, grâce à différents outils professionnels comme *XNA* ou *Unity*, a réussi à s'implémenter dans la sphère publique, et à répondre à une demande qui semblait croissante. De surcroît, les frontières entre les outils de développement sous licence et les outils publics s'amenuisaient au fur et à mesure de cette génération. C'est pourquoi nous pouvons désormais concentrer notre propos sur le moteur *Unreal Engine 3* puisque, d'après Joanna Lee, « Epic Games (...) [a publié] la troisième version de son moteur de jeu (...) en 2006 » (Lee 2016, 3 TDR)[507]. Néanmoins, la version grand public du moteur *Unreal Engine 3* a plutôt été distribuée en 2009, puisque « Unreal Development Kit (UDK), une version gratuite du logiciel qui offre (...) un accès aux (...) outils d'Unreal Engine 3 (...) [a été lancé] le 5 novembre » (GamesIndustry.biz 2009 TDR)[508] 2009. Celle-ci permettait de pouvoir utiliser le moteur *Unreal Engine 3* sans le besoin d'une licence professionnelle, et sans restriction aucune.

Néanmoins, une caractéristique importante du moteur *Unreal Engine 3* est la disponibilité de son code source. Un aspect qu'il est possible d'illustrer, d'après Morgan, à travers les jeux de la série *Batman: Arkham* (*ab*. 2009, PlayStation 3), puisque « Arkham Asylum et sa suite Arkham City (...) [qui] ont été initialement développés sur une version modifiée d'Unreal Engine 3 » (Morgan 2016 TDR)[509]. Naturellement, nous n'avons pas d'informations sur les modifications opérées dans cette version *d'Unreal Engine 3*. Toutefois, cela signifiait donc qu'il y avait un contrôle total sur les possibilités dans l'utilisation du moteur *Unreal Engine 3*, y compris une adaptation du moteur aux interactions des différents projets vidéoludiques.

Au regard de l'architecture du moteur *Unreal Engine 3*, on constate avec Klett que, « contrairement à d'autres grands moteurs 3D sur le marché (à l'exception de Gamebryo), les nouvelles versions du moteur Unreal ne sont pas développées

507. « Epic Games (...) [released] the third version of its game engine (...) in 2006 » (Lee 2016, 3).
508. « Unreal Development Kit (UDK), a free software release that provides (...) access to (...) Unreal Engine 3 toolset (...) [was launched] on Nov. 5 » (GamesIndustry.biz 2009).
509. « Arkham Asylum and its sequel Arkham City (...) [which] were originally built on a modified Unreal Engine 3 » (Morgan 2016).

à partir de rien » (Klett 2004, 64 TDR)[510]. La conséquence de cela est, pour les développeurs, une relative stabilité dans le fonctionnement du moteur, même si, comme le souligne également Klett, « Sweeney nous a informé que le code d'Unreal Engine 1 est véritablement minoritaire en ce qui concerne Unreal Engine 3 » (*ibid*. TDR)[511]. De surcroît, on peut également mettre en avant, grâce à Jason Bubsy, Zak Parrish et Jeff Wilson, que l'architecture du moteur « Unreal Engine contient plusieurs modules (...) [et] puisque tout est soigneusement modularisé, les titulaires de licence de la technologie Unreal Engine peuvent facilement personnaliser ou remplacer (...) les modules » (Bubsy, Parrish et Wilson 2009, 14 TDR)[512]. Une forme de modularité importante dans le cadre d'une flexibilité pour le développement d'un jeu vidéo, et qui implique qu'un tel moteur n'impose nullement une architecture particulière. Naturellement, la modification ou le remplacement d'un module requiert un travail plus important que l'utilisation d'un module déjà intégré au moteur.

Sur le plan de la programmation du *gameplay* ou de l'interactivité, nous pouvons mettre en exergue trois possibilités, toujours grâce à Bubsy, Parrish et Wilson (2009) : le langage *C++* (et l'accès au code source), seulement utilisable via une licence professionnelle *Unreal Engine 3*, le langage de *scripting UnrealScript* et le langage visuel *Kismet*. Nous ignorons donc la partie du *C++* puisque celle-ci est privée et concerne exclusivement *Unreal Engine 3*, et non pas *UDK*. Ainsi, et d'après Bubsy, Parrish et Wilson, *UnrealScript* peut être décrit comme étant un « langage de scripting qui permet aux programmeurs ou aux utilisateurs d'ajuster (...) tout ce que fait le moteur, sans modifier le code source du jeu » (Bubsy, Parrish et Wilson 2009, 18 TDR)[513]. De surcroît, on peut constater que le langage est conçu comme étant le plus haut niveau possible puisque « son créateur, Tim Sweeney, a déclaré que, lorsqu'il devait faire des compromis, il choisissait la simplicité plutôt que la vitesse d'exécution » (Bubsy, Parrish et

510. « unlike other major 3D engines on the market (with the exception of Gamebryo), newer versions of the Unreal engine aren't built from the ground up » (Klett 2004, 64).
511. « Sweeney tells us that code from Unreal Engine 1 is definitely in the minority in terms of Unreal Engine 3 » (Klett 2004, 64).
512 . « Unreal Engine contains several components (...) [and] since everything's neatly modularized, licensees of the Unreal Engine's technology can easily customize or replace (...) components » (Bubsy, Parrish et Wilson 2009, 14).
513. « scripting language that allows programmers or users to ajust (...) anything the Engine is doing, without touching the actual game source code » (Bubsy, Parrish et Wilson 2009, 18).

Wilson 2009, 18 TDR)[514]. Sweeney effectue également un comparatif avec un langage existant, puisqu'il indique que le langage de programmation « Java est très similaire à UnrealScript » (Epic Games 2012 TDR)[515]. Surtout, ce langage *UnrealScript*, spécialement conçu pour le *gameplay*, possède la « sensation et le style familier du code en C/ C++/ Java » (*ibid*. TDR)[516].

Au regard du langage visuel *Kismet*, Lee décrit celui-ci comme étant un « système de scripting visuel, qui a permis (...) de créer la logique du *gameplay* (...) sans avoir à se plonger dans l'écriture de code » (Lee 2016, 3 TDR)[517]. Cependant, ce système de *scripting* visuel est limité à un niveau donné, sans pouvoir exporter les *scripts* visuels en question vers un autre niveau au sein du moteur *UDK*. Pour utiliser notre outil de comparaison anachronique, on peut considérer que les scripts *Kismet* ne sont pas indépendants des *maps* comme le système visuel *Blueprints* du moteur *Unreal Engine 4*. En revanche, aussi bien dans le moteur *Unreal Engine 4* que la cinquième itération, on trouve les *Level Blueprints*, qui reprend justement le fonctionnement du langage *Kismet* et son lien avec les niveaux.

Par ailleurs, on constate avec Lee que l'interface *UnrealEd* est toujours présente, et celle-ci « fournit une interface visuelle (…) qui vous permet d'importer, d'organiser, d'éditer et d'ajouter des comportements/ interactions à vos *assets* du jeu » (Lee 2016, 9 TDR)[518]. De surcroît, nous pouvons également ajouter, sur la gestion des *assets* en temps réel, et d'après Bubsy, Parrish et Wilson, que le moteur « Unreal Engine 3 prend en charge le *Level Streaming* » (Bubsy, Parrish et Wilson 2009, 15 TDR)[519]. Une caractéristique relativement importante pour le développement de jeux vidéo intégrant le concept du monde ouvert.

Ainsi, cette vision historique du moteur *Unreal Engine 3*, ainsi que de sa version publique *Unreal Development Kit*, a permis de mettre en lumière ses multiples caractéristiques. Désormais, nous pouvons développer différentes théories

514. « its creator, Tim Sweeney, said that where he had to make tradeoffs, he chose simplicity over execution speed » (Bubsy, Parrish et Wilson 2009, 18).
515. « Java is very similar to UnrealScript » (Epic Games 2012).
516. « the familiar look and feel of C/C++/Java code » (Epic Games 2012).
517. « visual scripting system, allowed (...) to create game play logic (...) without having to delve into writing codes » (Lee 2016, 3).
518. « provides a visual interface (…) to enable you to import, organize, edit, and add behaviors/ interactions to your game asset » (Lee 2016, 9).
519. « Unreal Engine 3 supports Level Streaming » (Bubsy, Parrish et Wilson 2009, 15).

sémio-informatiques en exploitant ces informations sur le développement de jeux de la septième génération.

5-2. INTERACTANTS TECHNOLOGIQUES

Au regard de l'objet principal traité dans notre thèse, à savoir le développement de jeux vidéo en 3D temps réel, nous pouvons élaborer des concepts liés à l'utilisation des modèles 3D, réalisés à partir de différents logiciels. On nomme ces multiples outils DCC en anglais, ce qui signifie, pour reprendre la définition d'Henk Venter et Wilhelm Ogterop, « *digital content creation (...)* (par exemple Blender, Maya, 3ds Max, et Cinema 4D) » (Venter et Ogterop 2022, 155 TDR)[520], qu'on pourrait traduire par logiciel d'infographie 3D en français. Mais l'existence de la modélisation 3D est bien antérieure à la cinquième génération de consoles, puisque c'est le logiciel *Sketchpad* qui, selon Marco Hemmerling, « marqua la première fois où quelqu'un a réussi à afficher et à travailler avec l'infographie » (Hemmerling 2020, 62 TDR)[521]. Un outil qui a été développé et publié « en 1963, [par] l'informaticien américain Ivan Sutherland » (*ibid.* TDR)[522]. Et, plusieurs décennies après, *Sketchpad* est encore considéré comme étant, d'après Allan Blackwell et Kerry Rodden, « un des programmes informatiques les plus influents jamais conçus par un individu » (Blackwell et Rodden 2003, 3 TDR)[523].

En éludant l'historique gargantuesque des logiciels d'infographie, qui serait un hors-sujet pour notre recherche, nous pouvons simplement mettre en surbrillance des outils cités dans la documentation d'un moteur de jeu contemporain. Ainsi, le moteur *Unreal Engine* mentionne notamment les « application externe de modélisation 3D, telles que 3DS Max, Maya ou Blender » (Epic Games 2023, *Importing Static Mesh LODs Using FBX* TDR)[524].

On pourrait toutefois préciser que les logiciels *3DS Max*, *Maya* et également *Softimage* ont été utilisés dès la cinquième génération de consoles, à travers plusieurs exemples. Ainsi, pour le développement de multiples jeux sur la console

520. « digital content creation (...) (such as Blender, Maya, 3ds Max, and Cinema 4D) » (Venter et Ogterop 2022, 155).
521. « marked the first time anyone successfully drew and worked with computer graphics » (Hemmerling 2020, 62).
522. « in 1963, (...) [by] the American computer scientist Ivan Sutherland » (Hemmerling 2020, 62).
523. « one of the most influential computer programs ever written by an individual » (Blackwell et Rodden 2003, 3).
524. « external 3D modeling applications, such as 3DS Max, Maya, or Blender » (Epic Games 2023, Importing Static Mesh LODs Using FBX).

PlayStation, on retrouve des « utilisateurs de 3D Studio MAX (...) [tels que] *DreamWorks Interactive* (...) pour la modélisation et l'animation (...) de personnages et (...) d'environnements (...) afin de produire quatre nouveaux jeux interactifs » (Maestri *et al*. 1997, 24 TDR)[525]. Concernant la *Nintendo 64*, il est possible de mentionner, grâce à Alex Dunne, que « Nintendo a choisi Alias (un logiciel qui a été utilisé pour le [jeu] Super NES (…) Donkey Kong Country) comme système autorisé de création graphique pour (...) les jeux 64-bit » (Dunne 1995, 14 TDR)[526]. Précisons qu'*Alias* est l'ancien nom du logiciel *Maya*. On retrouve enfin l'utilisation de *Softimage* dans de nombreuses productions vidéoludiques, selon Dunne, puisque « Sega a choisi Softimage 3D comme outil de développement tridimensionnel officiel pour la nouvelle console Sega Saturn » (*ibid*. TDR)[527]. Ce dernier logiciel disparaîtra ensuite pendant la huitième génération de consoles, mais nous éludons cet aspect trop éloigné de notre recherche.

Néanmoins, notre objectif est plutôt d'évaluer l'existence de différences dans la manière dont sont gérés les modèles 3D dans un moteur. Pour cela, nous pouvons utiliser l'outil de comparaison anachronique de notre DLT, notamment avec le moteur *Unreal Engine 4*. Puis nous pourrons ensuite mettre en exergue la façon dont les interactants sont élaborés sur le plan technique, notamment avec les différentes classes dans ce même moteur. D'emblée, nous pouvons observer que ce même moteur va effectuer une subdivision entre une entité animé et une entité non-animé. Nous pourrions ainsi différencier un interactant humanoïde d'un interactant non-humanoïde. Plus précisément, la subdivision fondamentale des modèles 3D sur le moteur *Unreal Engine* est entre les *Static Mesh* et les *Skeletal Mesh*.

Les *Static Mesh* sont des « assets qui sont des unités de base utilisés pour créer les formes géométriques du monde (...) [et ces] modèles 3D sont créés

525. « users of 3D Studio MAX (...) [such as] DreamWorks Interactive (...) for modeling and animating (...) characters and (...) environments (...) to produce four new interactive game titles » (Maestri *et al*. 1997, 24).

526. « Nintendo selected Alias (whose software was used to create the Super NES (…) [game] Donkey Kong Country) as the authorized graphics development system for (...) 64-bit games » (Dunne 1995, 14).

527. « Sega chose Softimage 3D as the official three-dimensional development tool for the new Sega Saturn game platform » (Dunne 1995, 14).

avec des logiciels de modélisation externes » (Epic Games 2023, *Static Meshes* TDR)[528]. Un *Static Mesh* représente donc un modèle inanimé. Néanmoins, celui-ci peut quand même subir trois opérations mathématiques principales, par l'intermédiaire de ce qu'on appelle en anglais une *Transformation* : une *Translation*, une *Rotation* et un *Scale*. Nous pouvons traduire ces opérations par une translation, une rotation ou une modification d'échelle. En revanche, on différencie un *Skeletal Mesh* puisqu'il « contient le maillage visuel du personnage (...) et le squelette du personnage qui contient les données des *Bones* » (Epic Games 2023, *Skeletal Meshes* TDR)[529]. Nous considérons néanmoins qu'un *Static Mesh* peut tout autant être un interactant humanoïde qu'un *Skeletal Mesh*.

Nous pouvons désormais complexifier la notion d'interactant à travers les classes qui existent sur *Unreal Engine*, lesquelles sont exposées dans le tableau ci-dessous. Précisons d'emblée que, lorsque nous évoquons des classes telles que *Pawn*, *Actor* ou *Character*, nous nous concentrons surtout sur le langage *Blueprints*. Toutefois, les classes spécifiques en langage *C++* sont symbolisés par le préfixe originel de la classe *Unreal*, par exemple ***A**Pawn*, ***A**Actor* ou ***A**Character*.

Contrôleur	Interactant virtuel	Entités (Unreal Engine)
Player Controller	Humanoïde	*Pawn*
		Character
	Véhiculaire	*Pawn*
		WheeledVehiclePawn
	Immatériel	*Pawn*
		DefaultPawn
		SpectatorPawn

Tableau 32. Typologie des trois interactants virtuels sur *Unreal Engine 4*

528. « assets are the basic unit used to create world geometry (...) [and these] 3D models are created in external modeling applications » (Epic Games 2023, *Static Meshes*).
529. « contains the character's visual mesh (...) and the character's skeleton that contains bone data » (Epic Games 2023, *Skeletal Meshes*).

Cela étant, on peut observer une première subdivision en deux grands groupes : le contrôleur et les entités. Si nous énumérons uniquement la classe du *Player Controller*, on précise qu'il existe une classe similaire pour l'intelligence artificielle du jeu, nommée *AI Controller*, mais qui n'est pas pertinente dans le cadre de notre analyse. Quoi qu'il en soit, la documentation du moteur va définir un *Player Controller* comme étant des « *Actors* non-physiques qui peuvent posséder un *Pawn* (ou une classe dérivée de *Pawn* comme *Character*) pour contrôler ses actions (...) utilisés par des joueurs humains » (Epic Games 2023, Controllers TDR)[530]. Le rôle d'un *Player Controller* est donc de pouvoir posséder ou déposséder un *Pawn*, phénomène pouvant également être réalisé dynamiquement, c'est-à-dire en cours de *gameplay*. La documentation précise également que, « par défaut, il y a une relation un-à-un entre les *Controllers* et les *Pawns* (...) [ce qui] pourrait être ajusté (...) pour pouvoir contrôler plusieurs entités simultanément » (Epic Games 2023, Controllers TDR)[531].

Cela étant, nous pouvons constater que la classe *APawn*, qui hérite des deux classes *AActor* et *INavAgentInterface*, est la classe commune utilisable pour n'importe quel type d'interactant parmi les trois énumérés *supra*. Il s'agit d'une classe que la documentation va définir comme étant « la classe de base de tous les Actors qui peuvent être contrôlés par des joueurs ou des IA » (Epic Games 2023, Pawn TDR)[532]. Nous pouvons également illustrer les relations des classes ci-dessous.

530. « non-physical Actors that can possess a Pawn (or Pawn-derived class like Character) to control its actions (...) used by human players » (Epic Games 2023, Controllers).
531. « by default, there is a one-to-one relationship between Controllers and Pawns (...) [which] may need to be adjusted (...) to control multiple entities at once » (Epic Games 2023, Controllers).
532. « the base class of all Actors that can be controlled by players or AI » (Epic Games 2023, Pawn).

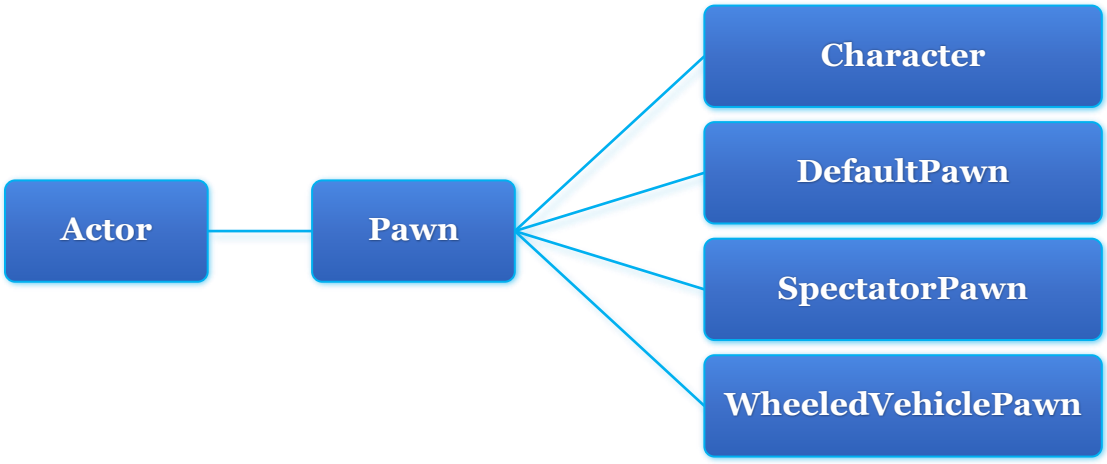


Figure 2. Classes des *Actors* dans Unreal Engine 4

Néanmoins, ces multiples classes ne sont nullement imposées par le moteur *Unreal Engine*, et semblent uniquement être des propositions utilisables dans certains contextes. L'interactant humanoïde ou l'interactant véhiculaire n'est donc pas conditionné par ces multiples classes, puisqu'il est possible d'utiliser une classe générique. De surcroît, au regard de la classe *APawn*, il est possible de constituer sa propre classe *APawn*, voire de modifier celle-ci dans le code source du moteur, pour l'adapter aux besoins d'un jeu particulier.

Ainsi, si nous pouvons considérer que certaines classes sur *Unreal Engine 4*, notamment *Character* ou *WheeledVehiclePawn*, sont spécifiques pour certains genres vidéoludiques, on souligne bien une possibilité d'utiliser des classes moins spécialisées, notamment celle de *Pawn*. C'est également le cas pour la classe *SpectatorPawn,* dérivées de la classe *Pawn* et qui semble appartenir au genre FPS, mais qui n'est nullement imposée par le moteur. Précisons, pour cette dernière classe et grâce à Wee Hoe Tan, que le mode spectateur semble en effet exister dans de nombreux FPS, et « permet à un joueur de regarder les parties d'autres joueurs dans un jeu » (Tan 2019, 199 TDR)[533].

Naturellement, les multiples indications que nous avons esquissées au regard du fonctionnement des classes *Player Controller*, *Actor*, *Pawn* ou encore *Character* sont des indications de classes standard pour le moteur. Il ne s'agit nullement de classes obligatoirement utilisables, puisqu'il est possible d'avoir *un Player Controller* inutilisé et d'implémenter les contrôles d'un interactant

533. « permits a player to watch how other players perform in a game » (Tan 2019, 199).

directement dans la classe *Pawn*. Surtout, cela dépend des multiples cas de figure, et Cedric Neukirchen indique bien qu'il est parfois préférable de « placer les *Input* spécifiques des *Pawn/Character* (...) dans les classes *APawn/ACharacter* et (...) les Input qui doivent fonctionner avec tous les personnages (...) dans le *PlayerController* » (Neukirchen 2023, APlayerController TDR) [534]. En substance, nous pouvons désormais préciser qu'il existe une définition concrète, autrement dit une implémentation, des trois types d'interactants dans le moteur *Unreal Engine 4*, notamment humanoïde (*Character*), véhiculaire (*WheeledVehiclePawn*) et immatériel (*DefaultPawn* et *SpectatorPawn*).

Néanmoins, ces multiples classes ne correspondent que peu ou prou à un genre précis, et restent relativement génériques, probablement en dehors de *SpectatorPawn*, plutôt proche du genre FPS. Surtout, la classe générique *Pawn* permet de pouvoir créer n'importe quel type d'interactant, voire sa propre définition de ce que doit être un interactant, sans rencontrer une quelconque limitation. De surcroît, la caractéristique de possession ou dépossession d'un *Pawn* implique la possibilité de créer une relation de complémentarité entre tous les interactants, c'est-à-dire de pouvoir posséder et déposséder tous les types d'interactants. Toutefois, un tel modèle implique également une certaine vision d'un univers virtuel, notamment pour nommer les classes d'une certaine manière ou encore pour octroyer des rôles à certains interactants, même s'il s'agit de rôles génériques. C'est pourquoi nous pouvons développer, dans la sous-section suivante, un concept non pas simplement de la vision d'un jeu vidéo, mais plutôt de la vision d'un monde virtuel, porté par un moteur de jeu.

534. « place Pawn/Character specific Input (...) into your APawn/ACharacter classes and (...) Input that should work with all Characters (...) into your PlayerController » (Neukirchen 2023, APlayerController).

5-2-A. VISION DÉVELOPPEMENTALE DU MONDE VIRTUEL

Les multiples implications technologiques du concept d'interactant, exposées précédemment, permettent de mettre en exergue d'éventuelles altérations dans la conception d'univers virtuels. Néanmoins, il s'agit d'une vision relativement partielle, voire partiale de la création d'univers. C'est pourquoi il nous semble important de développer le concept d'une vision développementale du monde virtuel. Une définition qui nous semble également découler de celui des paradigmes de programmation, mais qui se révèle être relativement conflictuel. En effet, le concept des paradigmes est rejeté par Saverio Perugini qui considère que, « si tous les langages sont des langages multi-paradigmes, alors le concept de paradigme de langage est désuet » (Perugini 2021, 18 TDR)[535]. Il propose plutôt la notion des styles de programmation, remplaçant *de facto* celui de paradigme, et probablement plus enclin à présenter une certaine flexibilité, du moins plus importante que pour la notion de paradigme.

Surtout, on peut également mettre en relief, grâce à Scott, le fait que ces multiples « distinctions entre les familles de langages ne sont pas nettement définies » (Scott 2016, 13 TDR)[536]. En revanche, notre objectif est d'aller plus loin que la notion de *style*, car nous considérons que l'impact de ces différents paradigmes va bien au-delà de l'organisation des multiples scripts, et particulièrement pour un moteur comme *Unreal Engine 4*. Nous pourrions formuler l'hypothèse que cela peut potentiellement altérer ou modifier substantiellement l'ensemble du programme, et plus encore l'élaboration de l'interactivité.

Néanmoins, il est important de rappeler que les paradigmes de programmation, ou les styles, ne constituent qu'une partie des multiples structurations que nous pouvons évoquer au regard d'un code informatique. En effet, il est également possible de mettre en exergue des aspects comme les *Design Patterns*, l'architecture complète d'un logiciel ou encore les différents algorithmes. Toutefois, nous ne pouvons pas nous permettre de définir ou

535. « if all languages are multi-paradigm languages, then the concept of language paradigm is antiquated » (Perugini 2021, 18).
536. « distinctions among language families are not clear-cut » (Scott 2016, 13).

d'approfondir ces notions sans s'éloigner du prisme de notre recherche et *in fine* de notre problématique. En revanche, il est intéressant d'observer que Ted G. Lewis considère les paradigmes comme étant « une manière de percevoir le monde, par exemple, un modèle idéalisé » (Lewis 2006, ch. 9.2 TDR)[537]. C'est justement cela qui pourrait nous indiquer que les paradigmes reflètent une certaine vision du monde, ce qui nous semble proche de l'hypothèse Sapir-Whorf.

Et, compte tenu de cette même hypothèse, nous pouvons d'abord indiquer avec Michel Aurnague que « c'est probablement chez Whorf que l'hypothèse initiale renommée "principe de la relativité linguistique" (et connue également sous l'appellation "hypothèse Sapir-Whorf") prend sa forme la plus radicale » (Aurnague 2004, 225). De surcroît, nous avons également la possibilité de définir celle-ci en expliquant, toujours selon Aurnague, que « Whorf pense que chaque langue renferme une théorie de la structure de l'univers (...) qui contraint notre vision du monde – la langue détermine (...) la culture plus qu'elle n'en est le reflet » (*ibid.*). Pour autant, et comme le précise également Aurnague, si « les positions de E. Sapir (...) [sont] moins claires (...) [il] paraît néanmoins adopter un point de vue plus modéré (...) [avec] une certaine séparation entre catégories linguistiques et catégories socioculturelles » (*ibid.*). Ajoutons que notre objectif est réellement d'éluder les multiples débats engendrés par cette hypothèse, voire les potentielles remises en question de celle-ci. Surtout, l'intérêt de cette corrélation entre l'hypothèse Sapir-Whorf et les paradigmes est simplement de mieux circonscrire la portée des paradigmes, voire du code informatique entier, sur la construction des univers virtuels.

En revanche, nous pouvons constater qu'il y a déjà eu une mise à proximité entre l'hypothèse Sapir-Whorf et les langages de programmation, puisque, comme le souligne Perugini, « certains idiomes de programmation ne peuvent pas être exprimés aussi facilement, voire deviennent inexprimables, dans certains langages alors qu'ils peuvent l'être dans d'autres » (Perugini 201, 5 TDR)[538]. C'est pourquoi le Professeur Chen va formellement identifier une potentielle corrélation, indiquant qu'il « adopte le concept controversé de (...)

537. « a way of viewing the world, e.g., an idealized model » (Lewis 2006, ch. 9.2).
538. « some programming idioms cannot be expressed as easily or at all in certain languages as they can in others » (Perugini 201, 5).

l'hypothèse Sapir–Whorf (...) et soutient l'idée selon laquelle les langues (humaines) déterminent, voire contraignent la cognition est pertinente dans les langages informatiques » (Chen 2018 TDR)[539]. Néanmoins, notre objectif est plutôt de rapprocher ce concept de notre définition des moteurs de jeu, pour justement vérifier si cette même hypothèse Sapir-Whorf est applicable dans un tel cadre.

Précisons que nous pouvons considérer qu'il s'agit plutôt du moteur de jeu qui influence, voire altère la vision du monde et l'interactivité, que du langage de programmation à lui seul. Nous pouvons étayer cette affirmation en utilisant plusieurs exemples de moteurs de jeu qui permettent, ou permettaient, l'usage de plusieurs langages de programmation. Par exemple, et d'après Penny de Byl, les versions antérieures du moteur *Unity* permettaient aux « développeurs d'utiliser trois langages interprétés : JavaScript, C# et Boo » (Byl 2014, ch. 2.1 TDR)[540]. Néanmoins, *Unity* n'a conservé que le langage *C#* parmi les trois mentionnés. De manière anachronique, nous pouvons également citer le moteur *Godot* qui « propose quatre langages de programmation de *gameplay* : GDScript, C# et, via sa technologie GDExtension, C et C++ » (Godot community 2024, *Scripting Languages* TDR)[541]. C'est également le cas pour *Unreal Engine 4* qui possède le langage visuel *Blueprints* ainsi que le *C++*, même si ces deux derniers peuvent être complémentaires, contrairement à ceux du moteur *Unity*.

Ces différents langages, s'ils introduisent des subtilités, ne modifient pas nécessairement la nature du moteur de jeu ou de l'univers virtuel qui en découlera. Plus encore, on trouve certains langages, notamment le *C++*, qui ont une compatibilité avec d'autres langages. Plus précisément, le *C++* possède, comme le rappelle Mitchell, une « rétrocompatibilité avec le C, en d'autres termes, la plupart du code C devrait compiler comme étant du C++ valide, sans nécessiter des changements significatifs du code » (Mitchell 2002, 337 TDR)[542]. C'est

539. « adopt the controversial concept of (...) the Sapir–Whorf hypothesis (...) and argue that the idea that (human) languages determine or even constrain cognition has its relevance to computer languages » (Chen 2018).
540. « developers to use three interpreted languages: JavaScript, C#, and Boo » (Byl 2014, ch. 2.1).
541. « offers four gameplay programming languages: GDScript, C#, and, via its GDExtension technology, C and C++ » (Godot community 2024, *Scripting Languages).*
542. « backwards compatibility with C, in other words, most C code should compile as legal C++, without requiring significant changes to the code » (Mitchell 2002, 337).

pourquoi nous pouvons surtout considérer que le concept de la vision du monde s'applique plutôt à un moteur qu'au langage de ce même moteur, et qu'il est régulé par l'ensemble des modules d'un moteur. En revanche, il sera capable de modifier la philosophie de la conception de jeu, c'est pourquoi on pourrait affirmer, grâce à Eleftheria Christopoulou et Stelios Xinogalos, que « les différents moteurs de jeu ont une philosophie différente en matière de développement de jeux, et visent une large gamme de besoins différents » (Christopoulou et Xinogalos 2017, 21 TDR)[543].

Nous développons ainsi le concept de la relativité technologique, car nous estimons que les outils de développement de jeux, créés *ex nihilo* ou *ex materia*, voire simplement acquis, vont imposer une certaine vision du monde ainsi qu'une définition des jeux vidéo. C'est pourquoi nous estimons également que chaque modification ou altération dans les outils de conception vont remettre en question, voire modifier la définition des jeux vidéo. Par exemple, nous estimons que les moteurs *Unity* ou *Unreal Engine* ne proposent pas la même vision du monde ni *in fine* la même conception des jeux vidéo. C'est pourquoi nous pouvons aller plus loin que Gregory lorsqu'il explique que les « moteurs de jeu sont généralement (...) spécifiques à un genre, [et] un moteur conçu pour un jeu de combat à deux (...) sera très différent d'un moteur pour un jeu en ligne massivement multijoueur (MMOG) » (Gregory 2019, 14 TDR)[544].

Ainsi, de la même manière qu'il est possible d'avoir plusieurs visions du monde, chaque moteur va imposer sa vision non seulement des jeux vidéo, mais aussi du genre principal qui est évoqué par Gregory. Car nous avons déjà observé en introduction que les genres sont peu circonscrits, et n'ont pas de définition propre (Arsenault 2011). Et, comme le rappelle Gregory, s'il existe « beaucoup de similitudes – [dans] tous les jeux en 3D, quel que soit le genre » (Gregory 2019, 14 TDR)[545], c'est également induit par l'imprécision intrinsèque des genres. C'est pourquoi un moteur n'est donc pas spécifique à un genre particulier, puisqu'un

543. « various game engines have a different philosophy on game development and aim at a wide range of different needs » (Christopoulou et Xinogalos 2017, 21).
544. « game engines are typically (...) genre specific, [and] an engine designed for a two-person fighting game (...) will be very different from a massively multiplayer online game (MMOG) engine (Gregory 2019, 14).
545. « great deal of overlap—all 3D games, regardless of genre » (Gregory 2019, 14).

genre ne correspond pas à des directives précises. En revanche, un même moteur va représenter sa propre vision d'un type de jeu ou d'un genre vidéoludique, et donc sa propre définition d'un genre. Par exemple, le moteur *Unreal Engine 3* ne représente potentiellement plus exactement le même modèle du FPS que ses itérations précédentes. De plus, ce même moteur possède une définition des FPS qui est différente d'autres moteurs spécialisés dans les FPS.

Plus encore, les notions de faisceaux virtuels et substrats virtuels permettent de vérifier si la vision développementale du monde émane d'un moteur en lui-même ou d'un jeu vidéo. Ainsi, dans le cadre d'un substrat virtuel, le moteur impose sa vision du monde de manière rigide, et l'univers virtuel n'a d'autre choix que de s'y conformer. Dans un tel cas de figure, le monde virtuel n'a strictement aucune possibilité d'exister en-dehors des limitations et contraintes du moteur. Or, le faisceau virtuel permet une plus grande liberté, puisqu'il s'agit de l'univers virtuel qui impose sa propre vision du monde, qui altère et modifie le moteur de jeu. Néanmoins, cette vision nécessite de pouvoir justement identifier différentes familles de signes interactifs dans le *gameplay* d'un jeu vidéo, voire dans la création des mécanismes interactifs d'un moteur. C'est pourquoi notre prochaine sous-section est justement consacré à la définition du niveau sémique du *gameplay*. Cela permettra de consolider la base théorique de notre vision développementale du monde.

5-2-B. NIVEAU SÉMIQUE DU GAMEPLAY

L'élaboration de cette nouvelle théorie de la vision développementale du monde virtuel a permis d'établir des fondations théoriques importantes pour élaborer un concept sous-jacent majeur, que nous nommons le niveau sémique du *gameplay*. L'intérêt de celui-ci est de pouvoir mettre en surbrillance non pas des signes interactifs isolés, mais plutôt des *familles* de signes interactifs, et de vérifier s'ils sont en concordance avec les limitations imposées par un moteur de jeu. Par ailleurs, cela va également permettre d'affiner d'autres concepts, notamment certains aspects théoriques des interactants virtuels, particulièrement au regard des multiples classes utilisables et du comportement de celles-ci dans un univers virtuel, possédant des familles de signes interactifs. De surcroît, nous allons associer la notion de sème interactif à celui de classe, afin de créer un alliage sémio-informatique pouvant rendre compte de particularités technologiques des signes interactifs.

Ainsi, afin de pouvoir présenter un tel concept, nous devons en premier lieu en restituer sa définition originelle. Pour cela, on constate avec Sarfati que « l'analyse sémique se propose de rendre compte des oppositions sémantiques élémentaires entre lexèmes, et suites de lexèmes mis en discours » (Sarfati 2020, ch. 7). Notre objectif, à travers ce niveau sémique, est donc de pouvoir isoler des unités interactives et de les analyser à travers trois concepts sous-jacents à notre niveau sémique interactif :

- Le sème technologique.
- Le sème technologique inhérent ;
- Le sème technologique afférent.

Tout d'abord, nous pouvons développer le premier concept, celui du sème technologique, en nous appuyant sur la définition d'un sème, impliquant de pouvoir isoler, selon Sarfati, des « unités minimales de signification » (*ibid.*). Sur cette base, nous pouvons considérer qu'il s'agit d'une unité minimale d'un signifié virtuel d'un signe interactif. Néanmoins, notre objectif est de réaliser une distinction fondamentale entre le signifié virtuel et le sème technologique. En effet, notre concept du sème a pour objectif de mettre au jour les multiples relations du signifié avec des classes en programmation, et de voir comment celles-ci peuvent s'exprimer à travers un seul et unique mécanisme interactif. Il

est donc substantiellement différent du signifié virtuel, qui ne concerne qu'un mécanisme interactif en lui-même, alors que le sème va vérifier sa hiérarchie avec des classes qui existent potentiellement dans un jeu ou un moteur de jeu.

C'est pourquoi nous pouvons complexifier cette notion à travers une distinction entre les sèmes technologiques inhérents et les sèmes technologiques afférents, en reprenant les définitions de Sarfati (2020). Celles-ci permettront de distinguer l'aspect définitoire des signes interactifs des connotations technologiques de celle-ci, notamment pour les aspects du moteur de jeu. Ainsi, un sème technologique inhérent se base sur le sème inhérent expliqué par Sarfati, « ou sème *dénotatifs* (...) [qui] confèrent à l'unité lexicale sa stabilité définitionnelle » (Sarfati 2020, ch. 7). Il s'agit donc du sens littéral d'un signifié du signe interactif dans un univers virtuel. Par exemple, le fait de conduire une *Banshee* et de rouler tout droit, de tourner à gauche ou à droite sont trois sèmes dans *Grand Theft Auto IV* (Rockstar Games 2008, PS3), et relève d'une potentielle classe conduite existant dans le jeu.

En revanche, le sème technologique afférent s'appuie sur la définition du sème afférent de Sarfati, « ou sèmes *connotatifs* (...) [qui] a pour particularité de relever (...) des normes sociolectales (...) ou idiolectales (...) » (Sarfati 2020, ch. 7). Naturellement, nous devons opérer une modification de ce concept pour l'adapter au développement de jeux vidéo, puisque l'analyse de normes, de surcroît sociolectales ou idiolectales, n'a aucune pertinence dans le cadre de notre étude. Nous pouvons en revanche concentrer nos analyses sur des normes développementales, liées à un moteur de jeu précis. Néanmoins, en utilisant notre théorie exposée *supra* du degré d'agnosticisme développemental, nous constatons que la plupart des univers virtuels des jeux vidéo commerciaux relèvent de technologies opaques. C'est pourquoi le sème technologique afférent va uniquement permettre de formuler des hypothèse sur un signifié, mais plutôt sur le plan de la substance développementale, autrement dit sur les scripts permettant de réaliser les signes interactifs bas niveau. Et, puisque Sarfati a abordé la notion de norme, nous pouvons également considérer que ce sème va intégrer des normes appartenant toujours à un moteur de jeu nommé et, de surcroît, public. On pourra donc relever des sèmes technologiques afférents en fonction des normes ou caractéristiques technologiques du moteur *Unreal Engine 4*.

Ainsi, pour reprendre un exemple précédent, nous pouvons d'abord considérer que la conduite de la voiture *Banshee* relève d'une classe proche du concept d'un *Pawn*, pour reprendre la terminologie du moteur *Unreal Engine* et réaliser une comparaison, hypothétique du moins. De celle-ci, nous pouvons observer qu'il y a une dépossession du personnage puis une possession de la *Banshee*. Nous pourrions attribuer des caractéristiques, autrement dit des propriétés et des méthodes à cette classe, et théoriser son comportement. L'intérêt théorique est donc majeur, puisque ce concept permet d'annihiler la limitation du signe interactif bas niveau. En effet, le sème technologique afférent devient applicable à des jeux vidéo commercialisés ou à notre corpus vidéoludique, autrement dit même en présence d'un univers virtuel intégralement crypté, puisque nous n'avons pas accès au moteur RAGE de *Grand Theft Auto IV* (Rockstar Games 2008, PS3).

Par conséquent, nous pouvons à présent identifier des signes interactifs bas niveau, même pour des jeux vidéo commercialisés. Cette notion de sème technologique afférent est d'ailleurs à l'origine de notre outil de comparaison anachronique. En effet, celui-ci permet de résoudre la problématique des technologies opaques. Néanmoins, le sème technologique permet d'analyser seulement une seule unité interactive, son intérêt est donc plutôt de permettre un regroupage de plusieurs sèmes technologiques. Nous pouvons désormais reprendre les multiples concepts dans le tableau ci-dessous.

Concept	Définition
Sème technologique	Unité minimale d'un signifié virtuel d'un signe interactif.
Sème technologique inhérent	Sens littéral d'un signifié du signe interactif.
Sème technologique afférent	Hypothèse sur un signifié du plan de la substance développementale et selon les normes d'un moteur de jeu.

Tableau 33. Concepts du niveau sémique interactif

Désormais, cette nouvelle nomenclature permet de mieux rendre compte des multiples aspects des mécanismes interactifs dans un même espace virtuel.

Celle-ci est donc adéquate pour réunir plusieurs familles. C’est cela qui permet désormais de développer le concept d’isotopie dans la sous-section suivante.

5-2-C. ISOTOPIE D'UN MONDE OUVERT

L'analyse de la création d'un univers virtuel d'un jeu vidéo a pour objectif de vérifier, dans notre recherche du moins, la combinaison des multiples signes interactifs. Le concept précédemment développé du niveau sémique interactif a permis de mettre en exergue des familles de sèmes technologiques, ou en termes profanes des *familles* d'interactions dans des univers virtuels. Néanmoins, notre objectif est d'aller au-delà de cette analyse partielle, afin de recouvrir un spectre complet, si possible, d'un seul et même univers virtuel. Un dessein relativement complexe, mais nous permettant d'appréhender la manière dont les multiples signes interactifs sont agencés, et d'évaluer la complexité du rapport entre eux. Afin d'atteindre cet objectif, nous allons désormais développer le concept d'isotopie interactive polymorphique. Précisons d'emblée que nous envisageons qu'un tel concept puisse permettre de mieux circonscrire et définir la structure des mondes ouverts.

Ainsi, le concept primaire de l'isotopie interactive polymorphique repose principalement sur la définition de l'isotopie sémantique. Sur ce point, nous pouvons préciser, en citant Rastier, que « le mot et le concept d'isotopie apparaissent pour la première fois sous la plume de A.J. Greimas » (Rastier 2022, 7), une notion empruntée à la physique ainsi qu'à la chimie. Néanmoins, nous allons ignorer les débats sémantiques et théoriques complexes sur cette notion d'isotopie, notamment au regard des multiples définitions illustrées par Rastier (2022). Au contraire, notre but est de reprendre une définition relativement restreinte mais également limpide. Ainsi, nous utilisons la définition d'une « *isotopie générale* [qui] consiste, à l'échelle macrotextuelle, de la répétition du même trait sémantique » (Sarfati 2020, ch. 7) pour l'adapter aux signes interactifs. Dans un tel cas, nous définissons l'isotopie interactive polymorphique comme étant la répétition de plusieurs sèmes technologiques au sein d'un même univers virtuel. Surtout, nous spécifions que ces différents sèmes doivent être en relation de complémentarité, c'est-à-dire qu'il doit être possible d'actualiser les différents signes interactifs dans une même séquence de *gameplay*, sans discontinuité.

Nous précisons que le terme polymorphisme est un concept informatique, que nous pouvons lier à l'isotopie en fonction du sens de celui-ci. Présent dans le

paradigme orienté objet d'un langage comme le *C++*, ce même concept de polymorphisme implique, comme le souligne Gregoire, que des « objets qui adhèrent à un ensemble de propriétés et méthodes peuvent être utilisés de manière interchangeable (...) [ce qui] exploite pleinement ce que l'héritage offre » (Gregoire 2021, 178 TDR)[546]. Autrement dit, le polymorphisme du *C++*, dans le contexte de l'orienté objet, implique une similitude de propriétés ou méthodes de plusieurs objets différents, permettant d'avoir une réponse, ou réaction, uniformisée ; une répétition semblable à celle d'un trait sémantique.

Nous pouvons surtout mettre en exergue, de manière plutôt succincte, ce concept d'isotopie interactive, notamment à travers la conduite dans *Grand Theft Auto IV* (Rockstar Games 2008, PS3). Celle-ci est composée de plusieurs sèmes technologiques identifiables et décomposables, qui sont présents tout au long du déroulement du jeu. Ainsi, la conduite des multiples véhicules relève d'une même isotopie, s'exprimant de différentes manières dans l'univers virtuel. On pourrait considérer une hypothétique classe conduite qui posséderait plusieurs classes dérivées, et où une même méthode, exploitant ici le polymorphisme d'un langage comme le *C++*, uniformiserait les interactions des différents véhicules. C'est pourquoi le concept d'isotopie interactive permet de définir de manière pertinente la combinaison des multiples signes interactifs dans un univers virtuel, et offrant une analyse plus adaptée d'un monde ouvert. Enfin, nous estimons que notre concept de l'isotopie d'un monde ouvert est suffisamment développé par rapport au cadre initial de notre thèse. Ainsi, nous éludons d'ajouter d'autres exemples qui ne pourraient en rien ajouter de nouveaux éléments pertinents.

546 . « objects that adhere to a standard set of properties and methods can be used interchangeably (...) [and it] truly takes advantage of what inheritance offers » (Gregoire 2021, 178).

5-3. SYNTHÈSE DE LA SEPTIÈME GÉNÉRATION

Le cinquième chapitre de notre recherche a permis de présenter la septième génération de consoles, qui débuta dès 2005 avec la distribution des consoles *Wii*, *PlayStation 3* et *Xbox 360*. Pendant cette ère technologique, nous avons observé une généralisation importante des moteurs AAA, mais également une démocratisation majeure des outils de conception pour le marché du développement indépendant.

5-1. SUPER-ORDINATEURS DE DIVERTISSEMENT

Dans la première section majeure, nous avons mis en avant les multiples moteurs AAA développés, avec une observation de la standardisation des moteurs ainsi que des outils de création de jeux vidéo, et particulièrement pour les moteurs commercialisés.

Dès **l'introduction**, nous avons focalisé notre attention sur une contextualisation de la distribution des consoles majeures de cette génération. Ainsi, la *Xbox 360* a permis de mettre en exergue l'utilisation du moteur *Unreal Engine 3* dans cette nouvelle génération (Hicks 2013). Puis la *PlayStation 3* était considérée comme étant un super-ordinateur de divertissement (CNET Australia 2008). On a également observé que l'intégration d'une connexion en ligne a permis l'émergence des boutiques en ligne, aussi bien sur consoles avec le *PlayStation Store*, le *Wii Shop Channel* et le *Xbox Live Arcade*, mais également *Steam* sur PC (Williams 2017). En outre, cela a également donné la possibilité de réaliser des mises à jour à distance des jeux commercialisés, phénomène qui a modifié l'industrie vidéoludique.

Et, dans la première sous-section, nommée **Généralisation des moteurs 3D internes**, nous avons d'abord exposé les difficultés imposées par l'architecture unique de la *PlayStation 3* pour le développement de jeux (Kent 2021 ; Scarpazza, Villa et Petrini 2007 ; Gregory 2019) mais également pour la conception de jeux sur plusieurs plates-formes (Graham 2018). Ensuite, nous avons également observé la standardisation des moteurs dans le développement de jeux, permettant de créer des jeux de plusieurs genres différents via un seul moteur (Klett 2004).

Et, dès la deuxième sous-section **Unification indépendante**, nous avons constaté la publication de plusieurs outils de développement pour les indépendants. En effet, *XNA* a vraisemblablement accéléré la démocratisation du développement pour les indépendant, étudiants ou amateurs (Sherrod 2008). Des moteurs importants ont également été distribués, notamment *CryEngine*, *Unity* ou encore *Unreal Development Kit*. Sur *PlayStation 3*, Sony a développé le moteur *PhyreEngine* pour les développeurs officiels (Middleton 2018).

Enfin, dans la troisième sous-section **Kit de développement protéiforme**, nous avons exposé les évolutions majeures du moteur *Unreal Engine 3*, mais également analysé la version grand public *Unreal Development Kit*. Ce moteur semble surtout posséder une certaine stabilité dans sa structure interne (Klett 2004) mais également une importante modularité (Bubsy, Parrish et Wilson 2009). Un même moteur pouvant permettre la création d'interactions via trois langages : le *C++*, *UnrealScript* et le langage visuel *Kismet* (Bubsy, Parrish et Wilson 2009 ; Lee 2016).

5-2. INTERACTANTS TECHNOLOGIQUES

Dans cette deuxième section majeure, nous avons développé de nombreux concepts sémio-informatiques, tout en renforçant nos théories précédentes. L'objectif principal était de pouvoir mieux circonscrire les signes interactifs dans les mondes ouverts.

Dès **l'introduction** de la deuxième section, nous avons précisé la notion d'interactant, notamment pour vérifier l'implémentation des trois types d'interactants dans le moteur *Unreal Engine 4*. Nous avons effectivement constaté que le moteur implémente un interactant humanoïde (*Character*), véhiculaire (*WheeledVehiclePawn*) et immatériel (*DefaultPawn, SpectatorPawn*). Il s'agit néanmoins de classes relativement génériques, ne correspondant à aucun genre précis en dehors de *SpectatorPawn*, assimilable aux FPS. La classe *Pawn*, plus générique encore, permet de créer n'importe quel type d'interactant, et nous avons également analysé la possession et dépossession des *Pawn*. Celle-ci permet d'envisager une relation de complémentarité entre les interactants.

Puis, dans la première sous-section intitulée **Vision développementale du monde virtuel**, nous avons constaté que le paradigme de programmation orienté

objet, un concept informatique relativement incertain (Perugini 2021 ; Scott 2016), était proche du concept d'une certaine vision du monde (Lewis 2006). Nous avons mis ce même concept en corrélation avec l'hypothèse Sapir-Whorf (Aurnague 2004), un lien déjà développé par le professeur Chen (2018). Néanmoins, notre objectif était surtout de l'appliquer aux moteurs de jeu, et non pas aux langages de programmation uniquement. Nous avons ainsi élaboré le concept de la relativité technologique, ou vision développementale du monde virtuel, où chaque moteur de jeu possède également sa propre vision du monde et *in fine* sa propre définition d'un genre vidéoludique.

Puis, dans la deuxième sous-section, nommée **Niveau sémique des mondes ouverts**, nous avons développé deux concepts basés sur des notions sémantiques : le sème technologique, subdivisé en sème technologique inhérent et sème technologique afférent. Ainsi, le sème technologique permet d'isoler l'unité minimale d'un signifié virtuel. Puis le sème inhérent représente le sens littéral d'un signifié du signe interactif, celui visible à l'écran. Or, le sème afférent est plutôt une hypothèse sur un signifié du plan de la substance développementale et selon les normes d'un moteur de jeu. Celui-ci permet donc de résoudre les problématiques engendrées par des analyses portant sur les technologies opaques, pour reprendre notre concept du degré d'agnosticisme développemental.

Enfin, dans la troisième sous-section, l'**Isotopie d'un monde ouvert**, nous avons réussi à mettre en exergue, grâce au concept d'isotopie interactive polymorphique, la combinaison des multiples signes interactifs dans un monde ouvert. Il s'agit d'un alliage sémio-informatique portant à la fois sur l'isotopie sémantique, c'est-à-dire la répétition d'un même trait sémantique (Sarfati 2020), et sur la polymorphie, des objets informatiques pouvant utiliser les mêmes méthodes et propriétés de manière interchangeable (Gregoire 2021). Cela a permis d'identifier différents sèmes technologiques présents tout au long du déroulement du jeu, et de vérifier s'ils possèdent des relations de complémentarité ou d'oppositions catégorielles.

6. HUITIÈME GÉNÉRATION : HYPERSTRUCTURE DES MOTEURS 3D

6-1. STANDARDISATION DES TECHNOLOGIES

L'année 2012 a marqué l'avènement de la huitième génération de consoles, entraînant une standardisation profonde du développement de jeux vidéo. Celle-ci débuta par le lancement de la *Wii U* de Nintendo en novembre 2012 aux États-Unis (Wardyga 2019, 418). Néanmoins, Reggie Fils-Aimé, à l'époque directeur de Nintendo of America, a bien mis en exergue qu'il existait « une certaine confusion quant au fait de savoir s'il s'agissait d'un nouveau système ou simplement d'une nouvelle manette pour la Wii existante » (Fils-Aimé 2022, ch. 15 TDR)[547]. Cette confusion générale a engendré un échec commercial, mais dont va néanmoins résulter, toujours selon Fils-Aimé, un « élément clef positif (...) [parce que] les joueurs nous ont dit qu'ils adoraient la possibilité de jouer à des jeux à la fois sur leur télévision grand écran et (...) la game pad de la Wii U » (*ibid*. TDR)[548]. Un autre aspect important, et toujours mis en exergue par l'ancien directeur de Nintendo of America, c'est l'absence de jeux vidéo indépendants sur la *Wii U*. Ceci en raison du fait que, comme le souligne également Fils-Aimé, « Nintendo s'est historiquement appuyé sur des outils développés en interne (...) mais l'industrie a changé (...) [avec] des entreprises fournissant des outils de développement essentiels comme Unity et Unreal Engine » (*ibid*. TDR)[549].

Ces deux constats exposés par Fils-Aimé permettront à Nintendo de développer une nouvelle console, la *Switch*, qui va remplacer la *Wii U* avec une distribution dès mars 2017 (Wardyga 2019, 399). Précisons, au regard de cette dernière console, qu'il existe des sources considérant celle-ci comme étant une console de neuvième génération. Or, au regard de sa technicité mais aussi des différents jeux qui sont disponibles sur cette même console, nous ne pouvons que l'affilier à la huitième génération de consoles. Dans tous les cas, nous pouvons observer, grâce à Hall, que la « console Nintendo Switch est plus ouverte

547. « some confusion whether this was a true new system or just a new controller for the existing Wii » (Fils-Aimé 2022, ch. 15).
548. « key positive element (...) [because] players told us they loved the ability to play games both on their big-screen television and (...) the Wii U game pad » (Fils-Aimé 2022, ch. 15).
549. « Nintendo had historically relied on internally developed tools (...) but the industry had changed (...) [with] critical development tool companies such as Unity and Unreal Engine » (Fils-Aimé 2022, ch. 15).

que jamais, et prend en charge des moteurs de jeu modernes comme Unreal et Unity » (Hall 2017 TDR)[550].

Au regard de la *PlayStation 4*, la distribution de cette console a commencé dès novembre 2014 (Wardyga 2019, 424). Surtout, dès les prémices de sa conception, il semblerait que Sony ait véritablement pris en considération les problématiques évoquées dans le chapitre précédent, notamment sur l'architecture de la *PlayStation 3* (Kent 2021 ; Scarpazza, Villa et Petrini 2007 ; Gregory 2019 ; Graham 2018). Cela notamment en modifiant l'approche utilisée pour concevoir l'architecture de la nouvelle console. En effet, une place plus importante encore a été attribuée aux développeurs de jeux vidéo pour la conception d'une console. C'est pourquoi Mark Cerny a été désigné comme étant l'architecte de la *PlayStation 4*, puisque, selon les propos de Wardyga, « aborder le nouveau matériel par un point de vue d'un développeur de jeux était une manière de corriger certaines difficultés auxquelles la société était confrontée avec la PS3 » (Wardyga 2019, 200 TDR)[551].

Cerny va également noter qu'il s'agit d'un phénomène qui va se distinguer des méthodologies du passé, puisque « impliquer la communauté de développement de jeux dans le processus de développement matériel est quelque chose (...) de très récent, historiquement parlant » (Cerny 2021, 3:14-3:20 TDR)[552]. Néanmoins, certains développeurs évoquent encore quelques difficultés avec l'architecture de la *PlayStation 4*. Ainsi, on peut évoquer l'absence de SSD, où Cerny indique bien que « nous avons reçu des demandes pour un SSD, depuis la PlayStation 4, en particulier de la part de Tim Sweeney (...), [qui] a affirmé que les disques durs constituaient un frein à l'évolution de l'industrie » (Cerny 2021, 1:13-1:23 TDR)[553].

Enfin, nous pouvons évoquer la nouvelle *Xbox One*, une console distribuée dès novembre 2013 (Wardyga 2019, 393). Concernant cette dernière, on peut constater, grâce à Sleep, que la « stratégie de Microsoft pour cette génération

550. « Nintendo Switch console is more open than ever, and supports modern game engines like Unreal and Unity » (Hall 2017).

551. « approaching the new hardware from a game developer's perspective was a way of correcting some of the difficulties the company faced with the PS3 » (Wardyga 2019, 200).

552. « bringing the game development community into the hardware development process is something (...) pretty recent, historically speaking » (Cerny 2021, 3:14-3:20).

553. « we've been getting requests for an SSD, all the way back to PlayStation 4, in particular, Tim Sweeney (...), [which] said, hard drives were holding the industry back » (Cerny 2021, 1:13-1:23).

était de transformer sa console en une unité de divertissement “tout-en-un”, d’où le nom » (Sleep 2022, 129 TDR)[554].

Sur le plan technique, et comme exposé par Wardyga, il est possible de mettre en exergue le fait que les « FLOPS (*floating point operations per second*) sont devenus plus importants que le nombre de polygones comme indicateur de la vitesse d’un processeur et des comparaisons entre consoles » (Wardyga 2019, 380 TDR)[555]. Néanmoins, si cette unité pour mesurer la puissance est plus précise, et par définition plus scientifique que le nombre de polygones, celle-ci ne pourrait en rien être pertinente pour jauger des possibilités interactives. Par ailleurs, les jeux nécessitent également une installation obligatoire, notamment sur *PlayStation 4* et *Xbox One* (Wardyga 2019, 396). Nous pouvons synthétiser les consoles dans le tableau ci-dessous, mais précisons que, contrairement aux chapitres précédents, nous ne pouvons pas indiquer les ventes des différentes consoles, certaines étant encore en production en 2024.

Consoles	Année de distribution
Sony PlayStation 4	2013
Nintendo Wii U	2012
Nintendo Switch	2017
Microsoft Xbox One	2013

Tableau 34. Ventes mondiales de la huitième génération

Au regard des multiples consoles évoquées, et surtout de l’élaboration de celles-ci, nous avons constaté que l’objectif était de faciliter le développement de jeux vidéo aussi bien AAA qu’indépendant. C’est pourquoi, dans les sections suivantes, nous analyserons ces deux aspects, qui semblent désormais fonctionner en synergie.

554. « Microsoft’s strategy for this generation was to turn its console into an “all-in-one” entertainment unit, hence the name » (Sleep 2022, 129).

555. « FLOPS (floating point operations per second) became more important than polygon counts as indicator for processor speeds and console comparisons » (Wardyga 2019, 380).

6-1-A. Réutilisation des moteurs internes

La démocratisation massive des moteurs de jeu internes lors des générations précédentes signifie que les studios de développement possèdent désormais des outils relativement complets, acquis grâce à l'expérience et l'expertise du passé. C'est pourquoi cette huitième génération va, à travers de nombreux exemples, illustrer une réutilisation plus importante des moteurs internes développés lors des générations précédentes. Ainsi, on peut d'abord illustrer ce phénomène avec le développement du jeu *Uncharted 4: A Thief's End* (Naughty Dog 2016, PS4), où le directeur Bruce Straley a « expliqué comment il ne répétera pas les erreurs passées de développer un nouveau moteur *from scratch* pour une nouvelle génération de consoles, comme il l'a fait (...) avec *Uncharted* pour la PS3 » (Reynolds 2013 TDR)[556].

Une situation que nous pourrions attribuer à l'augmentation substantielle de la puissance des consoles et autres types d'ordinateurs. Naturellement, Gregory indique que cette même puissance possède un impact sur les moteurs et la relation avec les genres, où « les distinctions entre les genres qui ont émergé en raison de préoccupations liées à l'optimisation commencent à s'estomper » (Gregory 2019, 31 TDR)[557]. En d'autres termes, on remarque que l'optimisation des jeux vidéo n'est plus aussi critique que pour, par exemple, la cinquième génération de consoles, qui nécessitait des technologies presque adaptées à chaque jeu. Ainsi, nous observons une certaine flexibilité des moteurs de jeu, et Romero va bien mettre en exergue le fait que, pour le *DOOM Engine*, « dans les années 1990, une équipe pouvait économiser seulement quelques mois avec notre moteur ; aujourd'hui, les moteurs permettent d'économiser des années aux équipes de développement » (Romero 2023, ch. 10 TDR)[558].

Plus encore, la commercialisation d'un nombre important de moteurs implique, selon Christopoulou et Xinogalos, que « de nombreux moteurs de jeu (...) partagent certaines fonctionnalités communes mais présentent également

556. « explained how it won't repeat past mistakes of developing a new engine from scratch for a new console generation, as it did (...) on Uncharted for the PS3 » (Reynolds 2013).
557. « differences between genres that arose because of optimization concerns are beginning to evaporate » (Gregory 2019, 31).
558. « in the early 1990s, a team might save mere months with our engine; now engines save development teams years » (Romero 2023, ch. 10).

de nombreuses différences » (Christopoulou et Xinogalos 2017, 21 TDR)[559]. Cela signifie *in fine* que de nombreux studios choisissent de se spécialiser dans un moteur plutôt qu'un autre, et que ces technologies ne sont nullement interchangeables. Mais, paradoxalement, on peut également constater qu'il existe, d'après Gregory, « beaucoup de similarités technologiques entre les genres, (...) il est donc de plus en plus possible de réutiliser la même technologie de moteur pour des genres très différents » (Gregory 2019, 31 TDR)[560].

Par ailleurs, si le langage *C++* reste majoritairement utilisé, il reste que, comme le souligne également Gregory, « plusieurs autres langages sont utilisés dans l'industrie vidéoludique – des langages impératifs comme le C ; des langages orientés objet tels que le C# et Java ; des langages de scripting comme Python, Lua et Perl » (*ibid.*, 105-106 TDR)[561]. Pour autant, nous pouvons préciser que la plupart des langages possèdent une implication différente du *C++*, puisque ce dernier langage est, dans la plupart des cas, également utilisé pour développer les moteurs de jeu. Toutefois, les statistiques portant sur l'utilisation des langages de programmation sont habituellement assez complexes à établir, voire impossibles, en particulier pour les moteurs internes. Nous ne pouvons donc nullement nous appuyer sur des informations factuelles, qui sont par essence indisponibles et invérifiables.

Enfin, nous estimons qu'il existe désormais une fusion des outils de développement de jeux vidéo amateurs et professionnels, où *Unreal Engine 4* et *Unity* sont utilisables dans les deux cas. En d'autres termes, ces moteurs sont adaptables aussi bien pour le développement d'un projet AAA que pour un jeu vidéo amateur. Néanmoins, on constate que les moteurs *Unity* et *Unreal Engine 4* sont réellement les deux moteurs principaux de cette huitième génération, et font l'objet de nos deux prochaines sous-sections.

559. « many game engines (...) share some common features but also have a lot of differences » (Christopoulou et Xinogalos 2017, 21).

560. « a great deal of technological overlap between genres, (...) it is therefore becoming increasingly possible to reuse the same engine technology across disparate genres » (Gregory 2019, 31).

561. « many other languages are used in the game industry — imperative languages like C; object-oriented languages like C# and Java; scripting languages like Python, Lua and Perl » (Gregory 2019, 105-106).

6-1-B. Expansion des moteurs de jeu publics

Dans l'industrie vidéoludique, le développement de jeux vidéo en 3D temps réel avec des moteurs de jeu publiquement disponibles oscille ainsi entre deux outils majoritairement utilisés : *Unity* et *Unreal Engine 4*. Mais, avant d'aller plus loin dans nos analyses, nous pouvons surtout mettre en surbrillance d'autres moteurs de jeu qui semblent néanmoins péricliter par la suite. Nous pouvons d'abord citer, grâce à Morgan Park, le moteur *Bitsquid* acquis « en 2014, [par] Autodesk (...) [qui] l'a renommé Autodesk Stingray, avant de mettre fin à son développement en 2018 » (Park 2024 TDR) [562] . Un moteur utilisant le langage *Lua* pour la programmation du *gameplay*. Néanmoins, ce même moteur ne semble pas avoir eu la possibilité de s'imposer dans le marché du développement de jeux vidéo, notamment face aux moteurs *Unity* et *Unreal Engine*.

Nous pouvons également mentionner la suppression du moteur *Blender Game Engine* pour la version 2.8 du logiciel DCC *Blender*, publiée en 2019. Nous pouvons préciser, grâce à Victor Kuller Bacone que ce moteur possédait un « système visuel de glisser-déposer pour créer des interactions de jeu basiques (...) [et] un langage de programmation, Python, (...) pour créer plus d'interactions complexes de jeu » (Bacone 2012, 8 TDR)[563]. L'unicité de ce moteur provient surtout du fait qu'il s'agissait d'un outil totalement unifié, à la fois pour concevoir les *assets* et les scripts de programmation. Cela semblait être un argument important pour faciliter le développement de jeux vidéo, et particulièrement pour des indépendants ou amateurs.

Il était donc possible de développer des jeux intégralement sur *Blender*, une opération impossible avec des moteurs actuels comme *Unreal Engine*, *Unity* ou *Godot*. Certes, si nous effectuons une comparaison anachronique avec *Unreal Engine 5*, il existe bel et bien des outils pour la modélisation 3D, voire l'animation ou la conception de textures, mais ceux-ci sont relativement simplistes, ne pouvant être envisageables pour la réalisation de modèles complexes. Et, au-delà de cette unicité, nous avons constaté qu'il n'existe, à notre connaissance, aucun

562. « in 2014, [by] Autodesk [which] renamed it Autodesk Stingray, then discontinued it in 2018 » (Park 2024).
563. « visual click-and-drag system to create basic game interactions (...) [and] a programming language, Python, (...) to create more complex game interactions » (Bacone 2012, 8).

jeu vidéo AAA ayant été développé via le moteur *BGE*. Ainsi, nous ne pouvons pas analyser ce même moteur de manière exhaustive. Néanmoins, nous pouvons formuler une hypothèse sur l'absence de l'utilisation du *BGE* pour des projets commerciaux : celle-ci pourrait être liée à des performance modestes du moteur, comparativement à d'autres outils plutôt populaires comme *Unreal Engine* ou *Unity*. Notons enfin, toujours au regard de ce moteur *BGE*, qu'il existe une version non officielle qui semble reprendre les objectifs de ce dernier : *UPBGE.* Cependant, celui-ci ne semble pas être utilisé dans le cadre du développement de jeux vidéo AAA, du moins lors de la rédaction de cette thèse.

Surtout, en ce qui concerne cette même popularité des différents moteurs évoqués, il est possible de s'appuyer sur les informations du site *SteamDB*, notamment via une page qui génère une liste des outils de développement de jeux les plus utilisés. Précisons toutefois que cet outil est utilisé de manière relativement anachronique, car nous nous appuyons sur des données datant de mars 2024. C'est ainsi que nous pouvons observer l'usage majoritaire des moteurs *Unity* et *Unreal Engine* pour un nombre considérable de jeux distribués sur *Steam*. Nous pouvons ainsi reproduire le tableau contenant les données sur les dix premiers moteurs ci-dessous.

SteamDB – Outils utilisés		
1	Unity	43 070
2	Unreal Engine	11 984
3	GameMaker	4 600
4	RPGMaker	3 000
5	PyGame	2 324
6	RenPy	2 260
7	Godot	1 227
8	XNA	1 005
9	Cocos	690
10	Adobe AIR	440

Tableau 35. Outils de développement listés par SteamDB (2024)

Au regard de ces résultats, nous pouvons constater une prédominance très importante du moteur *Unity*, puis une deuxième position majeure maintenue par le moteur *Unreal Engine*. En outre, Connie Loizos indique que le moteur *Unity* semble avoir été utilisé pour développer la moitié des jeux mobiles (Loizos 2017). Plus précis encore, Aaron Astle a souligné que les données de « data.ai ont identifié (...) [que] 71 des 100 plus grands jeux mobiles ont été développés avec au moins l'un des SDK de Unity » (Astle 2023 TDR)[564]. Autrement dit, l'usage de ce même moteur est extrêmement important dans le cadre du développement de jeux mobiles, même si cela est hors du cadre de notre recherche. Ainsi, nous pouvons illustrer ce phénomène d'utilisation massive de *Unity* dans de nombreux jeux mobiles par des exemples comme *Mario Kart Tour* (Nintendo 2019, Multi) ou encore *Subway Surfers* (Kiloo et SYBO Games 2012, Multi). Au niveau de la publication des jeux sur différentes consoles, nous pouvons citer *Ori and the Blind Forest* (Moon Studios 2015, Multi) ou encore *Cities: Skylines* (Colossal Order 2015, Multi).

Néanmoins, nous estimons qu'une analyse plus importante de ce même moteur n'est nullement pertinente au regard des informations que nous possédons déjà à travers les générations précédentes. Notre objectif étant simplement de répondre à la problématique, sans accumuler des informations superfétatoires. De surcroît, nous pouvons déjà exploiter nos données dans le cadre de l'élaboration de nos multiples théories. Ainsi, il est préférable de se concentrer davantage sur les caractéristiques majeures du moteur *Unreal Engine 4*.

564. « data.ai has identified (...) [that] 71 of the 100 biggest mobile games have been developed with at least one of Unity's SDK » (Astle 2023).

6-1-C. Démocratisation du moteur Unreal Engine

Deux ans après le démarrage de la huitième génération de consoles, la nouvelle itération du moteur *Unreal Engine* a été distribuée. Surtout, il ne s'agit pas d'une coïncidence ou d'un événement fortuit si chaque nouvelle version est liée à une génération de consoles, puisque Sweeney affirme, d'après Theo Valich, que le moteur « Unreal engine est réellement lié à un cycle de console » (Valich 2008 TDR)[565]. Il s'agit donc d'un procédé intentionnel, et cela implique également de constater des évolutions majeures pour chaque nouvelle itération du moteur en question.

Quoi qu'il en soit, cette quatrième version a été publiée, comme le rappelle Dave Tach, « en mars 2014 (...) [avec] un nouveau modèle économique : un abonnement mensuel de 19 $ » (Tach 2014 TDR)[566]. Surtout, cette nouvelle itération se différencie grandement de la version publique *Unreal Development Kit* de la génération précédente. En effet, plutôt que de concevoir plusieurs versions du moteur, notamment une première version destinée aux professionnels et une seconde pour les indépendants et amateurs, l'objectif était de créer une version totalement unifiée. Ainsi, celle-ci peut être exploitables aussi bien pour des studios AAA que des indépendants et amateurs.

De surcroît, cette quatrième version a permis un accès direct et immédiat au code source intégral du moteur, très majoritairement en langage *C++*. Cela permet de pouvoir effectuer des modifications du moteur dans le cadre du développement d'un jeu vidéo, et d'obtenir *in fine* un contrôle total sur le projet. Notons que nous ignorons dans notre étude les implications portant sur la licence du code source et la manière dont les modifications peuvent être apportées, comme indiqué dans nos chapitres précédents, même si nous avons mentionné que celle-ci était en *Source Available*.

Ainsi, au niveau des caractéristiques majeures du moteur *Unreal Engine 4*, nous pouvons nous concentrer sur celles impliquant la création du *gameplay*. Ici, nous pouvons évoquer deux possibilités : l'usage du langage *C++* et celle du langage visuel *Blueprint*. Si nous analysons le code source du moteur, nous

565. « Unreal engine is really tied to a console cycle » (Valich 2008).
566. « in March 2014 (...) [with] a new business model: a $19 monthly subscription » (Tach 2014).

pouvons également mentionner l'existence d'autres langages utilisés, notamment le *C#*, le *Python* et le *HLSL*, mais ceux-ci ne sont aucunement des langages pour la programmation du *gameplay*.

Ainsi, les deux langages utilisés pour le *gameplay* ont permis de remplacer *UnrealScript*, qui était présent dans les trois versions précédentes du moteur. Sweeney indique également la raison de l'abandon du langage *UnrealScript*, puisqu'il était question, d'après Valich, de le « remplacer (...) avec quelque chose de complétement nouveau, un système hautement *threadable* » (Valich 2008 TDR) [567]. Nous pouvons également ajouter que ce même moteur intègre également, d'après la documentation d'Epic Games, plusieurs « "micro-langages" (...) [et] le *Materials Editor* de UE4, le *Sequencer Tracks*, et le *AI Behavior Trees* sont tous des exemples de systèmes personnalisés pour stocker la logique du *gameplay* » (Epic Games 2021, *Balancing Blueprint and C++* TDR)[568]. Néanmoins, ces différents micro-langages informatiques ne représentent nullement des orientations ou limitations quelconques portant sur les genres vidéoludiques.

Par ailleurs, une vision néophyte sur ce moteur pourrait d'emblée opposer les deux langages présents dans le moteur *Unreal Engine 4*, autrement dit le *C++* et le *Blueprint.* Or, nous insistons sur le fait qu'ils sont élaborés pour fonctionner conjointement, comme le souligne Alex Forsythe, puisque « Unreal est conçu de telle manière que le C++ et les Blueprints soient véritablement complémentaires » (Forsythe 2021 TDR) [569]. Cette complémentarité implique également de mettre en exergue une subdivision de la programmation du *gameplay* en deux niveaux distincts, souvent décrite dans l'industrie vidéoludique et reprenant le concept des niveaux de langages informatiques exposé en introduction de cette thèse (Dixit 2011 ; Hugg 2018). Il s'agit ainsi du haut niveau et du bas niveau, que Forsythe (2021) va expliquer en termes de *scripting* de haut niveau et programmation de bas niveau. Plus précisément, il indique que le moteur *Unreal Engine* implique que le « C++ est naturellement mieux adapté pour

567. « replacing (...) with something completely new, a highly-threadable system » (Valich 2008).
568. « "micro languages" (...) [and] the UE4 Materials Editor, Sequencer Tracks, and AI Behavior Trees are all examples of custom systems for storing gameplay logic » (Epic Games 2021, *Balancing Blueprint and C++).*
569. « Unreal is designed in such a way that C++ and Blueprints are very complementary » (Forsythe 2021).

implémenter des systèmes de jeu bas niveau, et les Blueprints sont naturellement mieux adaptés pour définir des comportements et interactions haut niveau » (Forsythe 2021 TDR)[570]. Cela signifie surtout que les éventuelles contraintes techniques pouvant exister pour le langage visuel *Blueprint* peuvent être dépassées grâce au *C++*.

Cela étant, nous pouvons constater que le langage visuel *Blueprint* remplace directement *Kismet*, un langage visuel utilisé sur *UDK* et que nous avons mentionné dans le chapitre précédent (Lee 2016). Mais nous allons restreindre notre analyse sur le moteur *Unreal Engine 4* dans cette section, puisque nous avons déjà évoqué de nombreux aspects dudit moteur dans les chapitres précédents. Désormais, nous possédons suffisamment d'informations diachroniques sur le développement de jeux vidéo de la huitième génération pour développer de nombreuses théories sémio-informatiques, et surtout élaborer la structure ultime de notre recherche entière : l'hyperstructure 3D.

570. « C++ is naturally better-suited for implementing low-level game systems, and Blueprints is naturally better-suited for defining high-level behaviors and interactions » (Forsythe 2021).

6-2. HYPERSTRUCTURE 3D

Effectuer une recherche sur le développement de jeux vidéo, depuis les origines de la 3D temps réel jusqu'à la huitième génération de consoles, a permis l'élaboration de théories générales, hypercomplexes et polycéphales. Tout au long de ces chapitres, nous avons envisagé les multiples générations comme étant les symboles de véritables paliers technologiques, où les avancées annihilent au fur et à mesure les limitations techniques. Ce cheminement, intellectuellement et scientifiquement dispendieux, était toutefois nécessaire pour ériger des fondations solides de notre nouvelle structure sémiotique et informatique. Ainsi, notre hyperstructure 3D possède un objectif majeur : fournir un cadre théorique novateur pour les jeux vidéo, permettant une véritable représentation technologique et technique de notre hyperstructure 3D. Nous avons déjà présenté la définition de cette hyperstructure 3D lors du premier chapitre, très succinctement toutefois. Mais, pour fournir des informations plus exhaustives, nous pouvons préciser que nous concevons une véritable structure universelle de la création de jeux vidéo en 3D temps réel, mettant en exergue l'interactivité dans les univers virtuels. Outre l'objectif d'ordonner théoriquement le développement de jeux, cela permet de mettre en exergues les mécanismes interactifs, et de remplacer *de facto* la typologie des genres vidéoludiques, que nous jugeons obsolète au regard des outils contemporains de développement de jeux vidéo 3D.

Concrètement, cette structure est élaborée en tenant d'abord compte des facteurs technologiques intervenant dans les jeux vidéo. Ainsi, l'objectif est de replacer la technologie au cœur de cette structure théorique sémio-informatique. C'est la raison principale pour laquelle celle-ci va *in fine* intégrer et manipuler des notions techniques. Cela nous semble être le plus adéquat au regard des différentes analyses effectuées ainsi que de la nature intrinsèque des jeux vidéo, du moins en tenant compte de notre définition, où les jeux vidéo sont des logiciels informatiques. Précisons également que, nonobstant son exhaustivité, elle est surtout adaptée à l'analyse de la création d'un jeux vidéo, mais beaucoup moins pour étudier la conception d'un moteur de jeu ou d'une API 3D comme *Vulkan* ou *DirectX*. Nous considérons que cette dernière possibilité relèverait plutôt d'une méta-analyse, c'est-à-dire une analyse portant sur les outils de création en eux-

mêmes, et non pas sur des outils de création pour produire des jeux vidéo en 3D temps réel. Cela sortirait donc du cadre imposé par notre problématique et *in fine* notre recherche.

Dans tous les cas, nous présenterons notre conception de l'hyperstructure 3D, avec trois concepts majeurs qui permettent de décrire ses spécificités. De surcroît, ce triptyque conceptuel représente une structure multidimensionnelle, possédant trois dimensions fondamentales qui permettent de répondre directement à la problématique. Ces multiples dimensions reprennent l'ensemble des théories érigées jusqu'à présent. Néanmoins, l'objectif est surtout d'unifier, de consolider et de complexifier celles-ci, afin qu'elles puissent parfaitement répondre à notre problématique. Ainsi, voici les trois dimensions qui seront au cœur des trois sous-sections correspondantes :

- La dimension technologique ;
- La dimension interactive ;
- La dimension structurelle.

Précisons toutefois quelques aspects de ces multiples dimensions avant de les définir de manière exhaustive dans les sous-sections suivantes. Ainsi, la dimension primaire de notre structure, à savoir la dimension technologique, va permettre de définir le concept des moteurs 3D universalistes. Une théorie mettant en exergue les différents facteurs technologiques qui peuvent influencer le développement d'un jeu vidéo en 3D temps réel. L'objectif principal est de pouvoir vérifier les limitations potentielles et *in fine* l'existence des genres vidéoludiques. Ensuite, la dimension interactive incorporera différents concepts précisant les caractéristiques des signes interactifs, mais mettant également en exergue une absence des différentes limitations liées aux genres vidéoludiques.

Enfin, la dimension structurelle va s'appuyer sur les deux dimensions précédentes pour mettre en surbrillance le concept d'une nature structurelle. De surcroît, précisons que nos trois dimensions sont hiérarchisées et ordonnées logiquement. Par exemple, la dimension technologique représente les fondations de notre hyperstructure, sur laquelle la dimension interactive va s'appuyer. Nous pouvons désormais commencer par la présentation de la dimension technologique dans la prochaine sous-section.

6-2-A. Dimension technologique

La chronologie du développement de jeux vidéo esquissée tout au long de ces multiples chapitres a permis de développer, grâce à notre approche sémio-informatique, de nombreuses théories. Celles-ci permettent *in fine* de pouvoir mieux appréhender la conception des mécanismes interactifs et, de surcroît, de mettre en surbrillance les multiples métamorphoses du jeu vidéo. En effet, depuis ses origines incertaines dans les laboratoires des années 1960 puis des formes prototypiques de la 3D temps réel des années 1990, nous avons observé une avancée technologique majeure des moteurs de jeu jusqu'à la huitième génération de consoles ; presque une révolution alliée à une standardisation protéiforme. Il s'agit, en réalité, du décloisonnement progressif des moteurs de jeu et d'une généralisation de l'utilisation des moteurs par l'ensemble des acteurs de l'industrie vidéoludique. Néanmoins, nous considérons que l'appellation *moteur de jeu* n'est plus capable de recouvrir la réalité actuelle, et semble limitative quant à la portée réelle des outils de développement de jeux vidéo. Autrement dit, celle-ci est caduque, c'est pourquoi nous estimons qu'il est préférable de la remplacer par le concept d'un moteur 3D universaliste.

Cette même appellation reprend le terme *universel*, utilisé par Haigh et Ceruzzi, d'un « ordinateur [qui] pourrait également être appelé un solvant *universel*, (...) capable de faire disparaître une variété impressionnante d'autres technologies en intégrant celles-ci » (Haigh et Ceruzzi 2021, 3 TDR)[571]. Nous considérons que ce même concept de solvant universel, transposé à notre étude, est particulièrement intéressant au regard de l'évolution de plusieurs moteurs généralistes majeurs, puisque ceux-ci pourraient *in fine* remplacer de nombreuses technologies.

Ainsi, on pourrait citer *Unreal Engine* ou *Unity*, même si ce concept de moteur 3D universaliste pourrait être applicable à d'autres moteurs populaires. Les documentations officielles semblent également être rédigées de manière à s'éloigner du concept trop cloisonné de moteur de jeu. Pour illustrer notre propos, notons que la documentation du moteur *Unreal Engine* va définir celui-ci comme

571. « computer [which] might also be called a *universal* solvent, (...) by making an astounding variety of other technologies vanish into itself » (Haigh et Ceruzzi 2021, 3).

étant un « moteur en 3D temps réel conçu pour créer des jeux à la pointe de la technologie, des applications de visualisation en temps réel et des expériences interactives immersives » (Epic Games 2023, *Understanding the Basics: Foundational Knowledge* TDR)[572]. Quant à la documentation du moteur *Unity 2023.2 LST*, celle-ci va plutôt le définir comme étant un outil pour « créer des jeux en 2D et 3D, des applications et des expériences » (Unity Manual 2022.3 LTS, *Unity User Manual* TDR)[573]. La notion de moteur de jeu, c'est-à-dire d'un outil de développement exclusivement lié aux jeux vidéo, semble absente pour définir, voire circonscrire, le moteur *Unity*.

Naturellement, des moteurs internes ne peuvent prétendre à une telle définition. En effet, notre concept implique l'utilisation d'un moteur pour de nombreuses industries, s'étendant bien au-delà de celle des jeux vidéo ou d'un studio particulier. Or, la majorité, voire la totalité, des moteurs internes sont exclusivement utilisés par des studios de développement de jeux vidéo, du moins dans la plupart des cas. De surcroît, ils ne pourront nullement intégrer des fonctionnalités qui répondent à des besoins en dehors de l'industrie vidéoludique. En d'autres termes, notre définition implique plutôt un moteur disponible dans la sphère publique, car il s'agit du seul modèle capable de répondre à un spectre de besoins aussi large que possible. Et nous pouvons surtout préciser la dimension technologique du modèle théorique de notre concept du moteur 3D universaliste, qui va posséder les caractéristiques suivantes, basées sur l'intégration de nos théories antérieures :

- ► Une origine développementale (faisceau ou substrat) ;
- ► Une vision développementale du monde.
- ► Une abstraction tensive (intensité et extensité) ;

La première caractéristique principale est celle de l'origine du moteur, impliquant qu'un moteur peut ainsi être un substrat virtuel ou bien un faisceau virtuel. Le substrat implique donc un moteur totalement indépendant des jeux vidéo développés, avec des fonctionnalités relativement génériques et moins

572. « real-time 3D engine designed to create cutting-edge games, real-time visualizations, and immersive interactive experiences » (Epic Games 2023, *Understanding the Basics:* Foundational Knowledge).

573. « create 2D and 3D games, apps, and experiences » (Unity Manual 2022.3 LTS, Unity User Manual).

limitatives pour ledit moteur. Or, la version opposée est celle du faisceau, qui signifie qu'il existe potentiellement des éléments du moteur spécifiquement conçus pour un jeu précis.

Cette vision de l'origine développementale permet également de mettre en exergue le double aspect de notre hyperstructure, exposée lors du chapitre dédié à l'approche du DLT : une hyperstructure 3D interprétative et une hyperstructure 3D générative. Ainsi, l'origine développementale, dans le cadre d'une hyperstructure 3D interprétative, concerne l'influence potentielle des fonctionnalités du moteur pour le développement d'un jeu précis. Quant à l'hyperstructure 3D générative, c'est plutôt lié au développement du moteur en lui-même, permettant de savoir si son origine est liée à un jeu précis ou non. En l'occurrence, l'origine développementale du moteur 3D universaliste *Unreal Engine* est toujours liée au faisceau, puisque ce dernier est constamment utilisé pour le développement de jeux vidéo par *Epic Games*, le dernier en date et toujours en cours de développement en 2024 étant *Fortnite* (Epic Games 2017, PC). Or, l'origine d'un moteur comme *Unity* est toujours celle d'un substrat, car il n'est jamais réellement lié à un jeu précis et développé par *Unity Technologies*.

Au regard de la deuxième caractéristique, celle de la vision du monde, nous estimons qu'un moteur possède toujours une certaine vision du monde, et n'est à ce titre jamais purement générique, même s'il est décrit comme tel. C'est pourquoi notre théorie de la relativité technologique impliquait justement une certaine orientation et vision du monde. Précisions à nouveau que celle-ci n'est pas purement liée à un genre vidéoludique, mais plutôt à un modèle théorique d'un jeu, et nous considérons qu'un moteur est généralement optimisé pour ce même modèle théorique. Nous avons pris l'exemple du FPS pour les versions précédentes du moteur *Unreal Engine*, mais nous considérons aujourd'hui que *Fortnite* (Epic Games 2017, PC) représente le modèle théorique qui fait évoluer la vision du monde du moteur *Unreal Engine*, et cela depuis le début du développement de ce même jeu. Cela ne signifie nullement que le moteur est cloisonné à un genre particulier, ni même au *gameplay* de *Fortnite* (Epic Games 2017, PC), mais seulement qu'il est optimisé pour répondre aux besoins de ce même jeu. En revanche, la vision développementale du moteur *Unity* est plutôt conceptuelle, et le modèle théorique des jeux vidéo que propose *Unity* n'est pas lié à une œuvre existante, mais seulement à des besoins plus abstraits.

C'est pourquoi nous pouvons, en réalité, lier cette vision développementale du monde aux problématiques de l'optimisation d'un jeu vidéo pour un modèle précis et conceptuel d'un jeu. En effet, on considère que le développement d'un jeu vidéo qui tend à s'éloigner de cette vision développementale perdrait une certaine optimisation. C'est dans ce cadre que Gregory indique que « même les moteurs multiplateformes les plus polyvalents ne conviennent réellement que pour la création de jeux dans un genre particulier » (Gregory 2019, 12 TDR)[574]. Il est surtout question d'un manque d'optimisation pour créer des jeux dans un autre modèle que celui envisagé par le moteur de jeu, et non pas d'une impossibilité de création. Cela n'empêche nullement de s'éloigner du modèle théorique en question, et nous avons déjà mentionné cela, mais cela implique simplement une perte plus ou moins substantielle de l'optimisation. En substance, cela signifie que la vision développementale, c'est-à-dire le modèle théorique d'un jeu imposé par le moteur, peut potentiellement entrer en contradiction avec un genre vidéoludique spécifique ou un modèle de jeu particulier développé par un studio ou une équipe.

De surcroît, notre troisième caractéristique de la dimension technologique, à savoir l'abstraction tensive avec ses deux valences, mettent en surbrillance la complexité des langages de programmation ainsi que la diversité des modules ou systèmes dans un moteur de jeu. Sur le plan de l'intensité d'abstraction, nous avons estimé que celle-ci est élevée lorsqu'il existe un langage de programmation spécifique pour le moteur en question. Au regard du moteur *Unreal*, le langage visuel *Blueprint* est bien un langage propriétaire, spécialement conçu pour le moteur. Un tel langage nous semble être le plus adapté pour mettre en exergue des limitations potentielles dans la création du *gameplay*, et de surcroît celle des genres vidéoludiques. Nous avons ainsi constaté que, si certaines classes relèvent du genre FPS, celles-ci ne sont nullement obligatoires ou contraignantes, et il est toujours possible de créer un système personnalisé et adapté à des besoins différents. L'extensité d'abstraction, quant à elle, permet de mettre au jour les nombreux micro-langages présents dans le moteur *Unreal*,

574. « even the most general-purpose multiplatform engines are really only suitable for building games in one particular genre » (Gregory 2019, 12).

mais également les différents modules et systèmes dudit moteur, lesquels n'impliquent aucunement des contraintes liées aux genres vidéoludiques.

Grâce à l'élaboration de ces trois caractéristiques majeures de la dimension technologique de notre hyperstructure 3D, nous pouvons considérer une absence de contraintes liées aux genres vidéoludiques dans le moteur *Unreal Engine 4*. Et l'évolution technologique depuis la genèse de la cinquième génération de consoles a permis de s'affranchir de la nécessité d'une optimisation trop importante. C'est justement cet aspect qui a permis la création puis le développement de moteurs 3D universalistes, ne possédant pas de contraintes potentiellement rédhibitoires pour le développement d'un jeu en particulier. Néanmoins, il est nécessaire d'élaborer désormais la dimension interactive de notre hyperstructure 3D, afin de vérifier les éventuelles limitations des genres au-delà de l'aspect purement technologique.

6-2-B. DIMENSION INTERACTIVE

Grâce à la définition d'un moteur 3D universaliste exposé *supra* et intégrant une dimension technologique, nous pouvons mettre en avant une conceptualisation de l'interactivité plus proche des possibilités technologiques. D'abord, et cela dès l'introduction de cette recherche, nous avons décidé de séparer les jeux vidéo des autres formes de jeu, notamment en différenciant notre approche de celle des *Game Studies*. Cette même initiative, au départ réalisée à cause d'une simple incompatibilité de notre étude, a permis de mieux circonscrire les limitations induites par les outils de développement de jeux vidéo. De surcroît, nous avons constaté, à travers nos comparaisons anachroniques du moteur *Unreal Engine 4*, et également grâce à la conception de notre signe virtuel interactif, qu'il n'existe aucune réelle contrainte liée à la nomenclature des genres vidéoludiques dans les moteurs actuels. Néanmoins, notre objectif est désormais de mettre en exergue trois caractéristiques majeures pour définir la dimension interactive de notre hyperstructure 3D, qui reprennent les théories exposées dans les chapitres précédents, et qu'il est désormais possible de lister :

- ► Les signes virtuels interactifs ;
- ► Les sèmes technologiques ;
- ► Les interactivités générationnelles.

La caractéristique primaire de cette dimension interactive est naturellement le signe virtuel interactif, que nous avons développé dès le début des analyses portant sur la cinquième génération. Précisons néanmoins que la version interprétative de notre hyperstructure 3D intègre le signe interactif haut niveau, alors que la version générative utilise le signe interactif bas niveau. Ce signe a été développé afin de pouvoir repérer, éventuellement, l'existence ou l'expression de caractéristiques techniques spécifiques à certains genres vidéoludiques. Et nous avons constaté l'existence de mentions résiduelles du genre FPS dans certaines classes appartenant au moteur *Unreal Engine*, mais celles-ci n'étaient nullement des limitations ou contraintes. Il s'agissait de fonctionnalités utilisables seulement si nécessaire, induites à la fois par l'origine développementale, en l'occurrence celui d'un faisceau, mais également par la vision développementale, aujourd'hui liée au jeu *Fortnite* (Epic Games 2017, PC).

La deuxième caractéristique de notre dimension interactive est celle du sème technologique, qui représente un signifié virtuel et ses multiples relations avec des classes en programmation. L'objectif étant de vérifier comment des classes peuvent exister dans un seul et unique mécanisme interactif, du moins pour son signifié. Nous avons subdivisé cette unité en deux, le sème technologique inhérent et afférent, correspondant aux deux versions de notre hyperstructure 3D, respectivement interprétative et générative. De plus, le sème technologique afférent permettait de résoudre les problématiques du degré d'agnosticisme développemental ; raison pour laquelle nous n'avons pas intégré le degré d'agnosticisme dans notre hyperstructure, qui n'était intéressant que pour créer le sème afférent. Cela permet donc, pour l'hyperstructure interprétative, de pouvoir dépasser l'absence de connaissances sur le développement d'un jeu vidéo pour mieux spécifier les différentes relations entre les signifiés virtuels et des aspects techniques. Quant à l'hyperstructure générative, elle permet de réaliser une comparaison anachronique pertinente et pouvant résoudre l'absence de connaissances sur des moteurs propriétaires indisponibles pour le grand public. L'intérêt principal du sème technologique est donc de pouvoir in *fine* constater l'absence de contraintes liées aux genres vidéoludiques.

La troisième caractéristique majeure est celle de l'interactivité générationnelle, où nous reprenons d'abord les ancrages intragénérationnels, c'est-à-dire la présence des signes interactifs dans plusieurs jeux d'une même génération. Nous intégrons également les ancrages intergénérationnels, autrement dit l'existence avérée des signes interactifs dans de multiples générations. Le but de celle-ci est la réalisation de comparaisons entre les différentes générations, d'abord pour une observation de l'évolution des technologies. Ainsi, on relève plutôt des contraintes liées au matériel ou à l'absence d'outils standardisés, qui impliquaient de créer ses propres moteurs plutôt que d'utiliser des outils standards. En effet, la conception de son propre moteur permet d'optimiser le développement d'un jeu vidéo et *in fine* de s'adapter aux contraintes matérielles, contrairement aux moteurs standardisés comme *Unreal Engine*. Certes, il est possible de modifier les sources du moteur, notamment pour *Unreal Engine*, pour mieux s'adapter aux contraintes. Néanmoins, l'intérêt de l'utilisation d'un moteur de jeu est justement de s'affranchir, lorsque c'est possible, de la création, voire la

modification d'un moteur. Phénomène que nous avons exposé grâce à Pedersen (2003).

Au regard des multiples éléments exposés, nous avons constaté que les différentes théories et analyses ne permettent nullement de mettre en exergue une quelconque limitation de l'interactivité induite par les genres vidéoludiques dans les moteurs 3D universalistes. Les moteurs permettant de développer des jeux vidéo dans la plupart des genres ont été exploitables dès la sixième génération de consoles, comme nous l'avons illustré avec *Unreal Engine 2* ou encore les deux versions de *RenderWare*. C'est pourquoi nous pouvons désormais développer les linéaments de la dernière dimension de notre hyperstructure 3D dans la prochaine sous-section : la dimension structurelle.

6-2-C. DIMENSION STRUCTURELLE

Au-delà de la dimension interactive d'un moteur 3D universaliste, nous pouvons désormais reprendre certains concepts des chapitres précédents, qui sont au cœur de cette dimension structurelle. L'objectif principal de celle-ci serait de pouvoir mettre en relief la structure du monde virtuel dans le processus de développement de jeux vidéo. Nous pouvons ainsi présenter ces trois concepts :

- Un triptyque des interactants ;
- Une isotopie interactive polymorphique ;
- Une nature structurelle.

Désormais, nous estimons que la caractéristique primaire de la dimension structurelle est celle de l'interactant. En effet, l'interactant est une entité ou un élément qui permet de pouvoir entrer en interaction dans un monde virtuel. Nous avons développé une nomenclature de trois interactants virtuels, notamment l'interactant humanoïde, véhiculaire et immatériel. Sur le plan de l'hyperstructure 3D générative, nous pouvons constater que le moteur *Unreal Engine 4* permet de basculer d'un interactant à un autre grâce à la possession et dépossession. De surcroît, nous avons exposé dans la section des interactants technologiques différentes classes, notamment la classe principale *Pawn* qui est utilisable pour tous les interactants possibles. Cette même classe illustre bien une absence de contraintes spécifiques liées aux genres vidéoludiques.

La deuxième caractéristique porte sur l'isotopie interactive polymorphique, qui représente la combinaison des multiples sèmes technologiques dans un monde virtuel. Car, contrairement aux sèmes qui relèvent plutôt d'interactions isolées, l'isotopie interactive vérifie plutôt les relations entre les différentes interactions. Un interactant peut ainsi réaliser plusieurs signes interactifs, mais la relation entre ces multiples sèmes, et la combinaison de ceux-ci avec d'autres signes pour former différents systèmes, est semblable aux micro-langages d'un moteur de jeu. De surcroît, l'intérêt de cette théorie étant de pouvoir vérifier si des limitations liées aux genres vidéoludiques peuvent s'exprimer, pour un moteur de jeu, dans d'éventuelles relations de complémentarité. Il s'agit justement d'une potentielle contrainte majeure, car l'absence de relation de complémentarité possible entre plusieurs sèmes indique justement une limite potentielle liée à des genres vidéoludiques. Or, nous n'avons pas constaté des contraintes techniques

pouvant être rattachées aux genres vidéoludiques et entravant la combinaison de plusieurs sèmes technologiques, du moins pour les moteurs comme *Unreal Engine 4*.

La troisième et dernière caractéristique majeure est celle de la nature structurelle, une notion développée en même temps que les faisceaux et substrats virtuels. Celle-ci permet de mettre en exergue les propriétés et *in fine* la structure d'un simulacre ou du monde virtuel dans lequel les interactants peuvent évoluer. Mais, surtout, cette même nature est conditionnée par les spécificités du moteur de jeu utilisé pour développer un jeu vidéo et l'exécuter. Et c'est justement la subdivision en deux catégories exclusives, un hapax structurel ou une instance structurelle, qui implique de pouvoir mettre au jour les caractéristiques d'un monde virtuel.

6-3. SYNTHÈSE DE LA HUITIÈME GÉNÉRATION

Ce sixième et ultime chapitre de notre recherche était d'abord consacré à la huitième génération de consoles, notamment déclenchée par la distribution de la *Wii U* dès 2012, puis de la *PlayStation 4*, de la *Xbox One* ainsi que de la *Switch*. Mais, surtout, celui-ci a permis de présenter notre structure sémio-informatique inédite, l'hyperstructure 3D, avec ses trois concepts majeurs ainsi que ses trois dimensions fondamentales. Une structure qui a *in fine* permis de modifier substantiellement la définition même des jeux vidéo, mais également de mettre au jour la structure du développement des logiciels interactifs en 3D temps réel.

Notons également que ce récapitulatif ne possède pas de synthèse de l'hyperstructure 3D, puisque celle-ci permet déjà de récapituler de nombreuses théories disséminées dans cette recherche. Réitérer nos propos une nouvelle fois semblait donc trop répétitif, notamment au regard de la nature intrinsèque de notre hyperstructure.

6-1. STANDARDISATION DES TECHNOLOGIES

Cette première section a permis de mettre en exergue une standardisation plus importante encore des outils de développement dans l'industrie vidéoludique, mais également de la démocratisation de moteurs exploitables aussi bien pour des professionnels AAA que des indépendants et des amateurs.

Dès **l'introduction**, nous avons évoqué le lancement de la *Wii U* en 2012, un échec commercial imputable à au moins deux facteurs : la dénomination de la console et l'absence de jeux indépendants (Fils-Aimé 2022). Ainsi, l'appellation a induit en erreur des acheteurs potentiels, pensant qu'il s'agissait d'une manette *Wii*. De plus, des moteurs comme *Unity* ou *Unreal Engine* n'étaient pas accessibles sur *Wii U*, entravant la conception de jeux indépendants. Deux aspects qui seront vraisemblablement corrigés sur la console suivante de Nintendo, la *Switch,* dès 2017 (Wardyga 2019, 399). Nous avons également constaté que, pour la console *PlayStation 4* distribuée en 2014 (Wardyga 2019, 424), Sony a modifié sa méthodologie de conception des consoles. En effet, il était désormais question d'impliquer les développeurs de jeux, notamment pour éviter une console trop complexe à programmer (Cerny 2021). Notons également que, pour la distribution de la *Xbox One* dès 2013 (Wardyga 2019, 393), Microsoft

avait pour stratégie de transformer sa console en un divertissement tout-en-un (Sleep 2022, 129).

Puis la première sous-section **Réutilisation des moteurs internes** a illustré une réutilisation majeure des moteurs propriétaires internes développés lors des générations précédentes. Le facteur principal de cette réutilisation est l'augmentation substantielle de la puissance des machines, réduisant ainsi la nécessité d'optimiser les jeux développés et les distinctions techniques entre les genres (Gregory 2019, 31), voire entre les générations. Des moteurs de jeu qui permettaient d'économiser des années de développement, contrairement aux années 1990 où cela pouvait seulement accélérer le processus de création de quelques mois seulement (Romero 2023, ch. 10). De surcroît, on a constaté une fusion des outils de développement pour la création de jeux amateurs, indépendants et professionnels avec les moteurs *Unreal Engine 4* et *Unity*.

Ainsi, dès la deuxième sous-section, intitulée **Expansion des moteurs de jeu publics**, nous avons observé la publication d'autres moteurs de jeu qui semblaient péricliter par la suite. Par exemple, le moteur *Bitsquid*, renommé ensuite *Stingray*, mais également le *Blender Game Engine*. Surtout, la base de données *SteamDB* (2024) indique que les moteurs *Unreal Engine* et *Unity* semblent réellement majoritaires, même si les statistiques dans l'utilisation des moteurs est toujours relativement incertaine. De surcroît, le moteur Unity semble avoir été utilisé pour développer la moitié des jeux mobiles (Loizos 2017), et 71 des 100 plus grands jeux mobiles (Astle 2023), impliquant une utilisation presque majoritaire sur le marché du mobile.

Enfin, la troisième sous-section, nommé **Démocratisation du moteur Unreal Engine**, a permis de mettre en exergue la publication du moteur *Unreal Engine 4* dès 2014 (Tach 2014), une version identique pour tous les utilisateurs du moteur. Celle-ci permet également de consulter le code source librement. Une situation distincte du moteur *UDK* de la génération précédente, puisque *UDK* était notamment destinée aux indépendants et amateurs. De surcroît, avec le moteur *Unreal Engine 4*, la création du *gameplay* était réalisée grâce au langage *C++* ainsi qu'au langage visuel *Blueprint*. Ces deux langage sont conçus dans une relation de complémentarité, avec le *C++* représentant la programmation bas niveau et le *Blueprint* le *scripting* haut niveau (Forsythe 2021).

7. CONCLUSION :

IMMUABILITÉ TECHNOLOGIQUE

7-1. TECHNOLOGISATION MULTIDIMENSIONNELLE

À travers nos pérégrinations tumultueuses dans les dédales complexes et imprévisibles du développement de jeux vidéo en 3D temps réel, nous avons développé un conglomérat de théories bicéphales en sémiotique appliquée et en informatique. Mais c'est surtout grâce à l'élaboration de notre approche sémio-informatique inédite, le Développement Logiciel Tridimensionnel (DLT), que nous avons réussi à modéliser une structure unifiée de la création des univers virtuels. Pourtant, le chemin était particulièrement sinueux et difficile, avec une problématique intergénérationnelle et presque sibylline pour un néophyte, interrogeant l'existence de limitations liées aux genres vidéoludiques dans les outils de conception de jeux vidéo en 3D temps réel.

De surcroît, l'objectif principal de cette recherche était de vérifier si cette même typologie, dont la genèse est située au début des années 1980, était liée à des contraintes technologiques ou seulement à des conventions arbitraires. Des conventions hypothétiquement imposées par des développeurs, des journalistes de jeux vidéo ou encore des joueurs, mais dénuées de réalité sur le plan technologique. À ce titre, nous avons élaboré une hypothèse de départ qui mettait déjà en avant une conventionnalité apparente des genres vidéoludiques, vraisemblablement sans rapport avec des limitations techniques de l'interactivité.

D'emblée, nous nous attendions à la simple confirmation de notre hypothèse de départ dans cette conclusion, à réitérer nos propos et à employer une prose ornée d'arguments. Or, le résultat final nous semble bien trop complexe pour esquisser une conclusion si simpliste. Pourtant, les genres vidéoludiques sont bel et bien conventionnels, c'est-à-dire qu'ils ne sont pas liés à de véritables limitations technologiques, du moins lors de la création d'interactions via un moteur de jeu et pour les moteurs actuels. Ils sont plutôt définissables par les multiples jeux vidéo, et au fil de l'évolution de la technologie des moteurs. De surcroît, les limitations liées aux genres vidéoludiques semblent désormais peu perceptibles dans les moteurs de jeu contemporains. *In globo*, un moteur de jeu n'est pas un simple logiciel de création de jeu vidéo en 3D temps réel générique. En effet, il permet surtout de créer des mondes virtuels interactifs, proposant en réalité une certaine vision du monde, un prisme à travers lequel on peut concevoir

certains modèles de mondes virtuels. C'est pourquoi nous avons élaboré notre propre définition d'un moteur 3D universaliste qui encapsule cette conception, où on perçoit désormais ces outils comme subjectifs et partiaux.

Par ailleurs, l'objectif initial de cette thèse était également de construire une passerelle entre deux univers distincts : le monde scientifique, empreint d'une rigueur intellectuelle et méthodologique importante, et le monde professionnel du développement de jeux vidéo. Surtout, l'étude des *Game Studies*, en plein essor depuis les années 1990, a permis de dresser un constat limpide mais majeur : ce domaine académique n'intègre pas pleinement des problématiques du développement de jeux vidéo, empêchant ainsi de saisir l'essence des jeux vidéo, d'appréhender la réalité des univers virtuels, bien au-delà d'analyses apparentes.

Or, l'impact sociétal des jeux vidéo n'est plus à démontrer, et son importance est majeure dans les enjeux sociaux et économiques du XXI^{e} siècle. C'est la raison pour laquelle notre approche du DLT, loin d'être une ramification des *Game Studies*, repose sur un objectif scientifique précis et concret : une analyse technologique des jeux vidéo reposant non pas sur des éléments purement discursifs, mais sur des aspects techniques, autrement dit des moteurs de jeu. Par ailleurs, cette même approche a principalement permis de réaliser une étude diachronique couvrant plusieurs générations de consoles, depuis la cinquième jusqu'à la huitième incluse, afin d'en analyser les aspects techniques. Nous pouvons désormais synthétiser l'étude de toutes ces générations, ainsi que des multiples observations effectuées en introduction de cette thèse doctorale.

7-1-A. PRÉMICES DES MOTEURS TRIDIMENSIONNELS

En reprenant l'histoire des jeux vidéo, on observe que les trois premières générations sont juchées d'œuvres relativement primitives et abstraites, depuis la toute première console, la *Magnavox Odyssey* distribuée dès 1972, jusqu'à la machine *Atari 2600* qui accélérera le crash du jeu vidéo de 1983. L'élément central, au cœur de notre étude et dont la genèse semble être située vers la fin de la troisième génération de consoles, c'est justement celui des genres vidéoludiques. En effet, peu après l'émergence des premiers jeux et consoles, l'industrie vidéoludique a eu deux crises, d'abord un léger krach en 1977 puis une crise majeure qui avait le potentiel de faire péricliter un domaine naissant : le krach de 1983. Les genres semblaient, après cette crise et au regard du croisement des sources, une forme de réponse partielle pour ordonner un marché vraisemblablement trop chaotique, et prémunir une éventuelle crise ultérieure. Cela pouvait également représenter un critère dans l'établissement de normes de qualité, d'abord par Nintendo avec le *Seal of Quality*, puis par l'ensemble des acteurs de cette industrie, en particulier sur les consoles de jeu.

Certes, il n'est pas possible de déterminer avec exactitude une éventuelle corrélation entre le Krach de 1983 et l'émergence d'une théorisation des jeux vidéo par Crawford (1984), incluant cette nomenclature presque darwinienne. Néanmoins, une nomenclature portée sur l'interactivité ou le *gameplay* semblait réellement pouvoir guider un marché vidéoludique encore incertain. Celle-ci a naturellement évolué à travers le temps, mais semble être traversée par un paradoxe majeur : les genres vidéoludiques sont basés sur des définitions génériques, formulées de manière très laconique, et dont la distinction n'est pas toujours évidente. Dès lors, l'objectif de cette recherche était de vérifier si les descriptions primitives sur les genres étaient esquissées en fonction de contraintes technologiques d'antan ou simplement de conventions. Cela exigeait donc vérifier une hypothétique existence de contraintes dans les moteurs de jeu qui ressembleraient, peu ou prou, aux genres vidéoludiques.

C'est pourquoi nous avons débuté, au cœur de notre thèse, une analyse diachronique, en évoquant succinctement la quatrième génération de consoles, qui débuta en 1987 mais où nous avons simplement évoqué la console *Super Nintendo*. Pour développer des jeux vidéo sur celle-ci, il était nécessaire d'utiliser

la famille des langages *assembleurs*. Et, par nature, un langage assembleur ne pouvant être assemblé que pour une famille spécifique de CPU. Autrement dit, le développement de jeux vidéo était réalisé sur mesure, spécifiquement pour cette console et sans possibilité de porter aisément le code dans une autre console. C'est pourquoi le concept de moteur de jeu était encore très balbutiant, voire presque inutile. Cela non seulement à cause des contraintes naturelles des langages *assembleurs*, mais également au regard de la faible complexité des jeux à l'époque, en comparaison à celle d'aujourd'hui.

Par la suite, la cinquième génération de consoles a débuté au milieu des années 1990, avec trois consoles majeures : la *Sony PlayStation*, la *Sega Saturn* ou encore la *Nintendo 64*. Il s'agit de l'ère des premiers jeux en 3D temps réel, autrement dit de la genèse de cette nouvelle technologie qui ne possédait aucune uniformisation ou harmonisation. C'est pourquoi, sur le plan technique, nous avons observé une standardisation importante de plusieurs outils dans le développement de jeux, principalement avec trois technologies majeures : le langage de programmation *C*, les API comme *OpenGL* ou *DirectX*, et enfin les moteurs de jeu. Surtout, la première *PlayStation* a hypothétiquement introduit une nouvelle approche dans le développement de jeux, notamment avec une utilisation directe d'un ordinateur pour la création, mais également le développement multi-plateforme de jeux vidéo.

De surcroît, la création des premiers moteurs de jeu comme le *Doom Engine*, les deux *Quake Engines* ou *Unreal Engine,* était fortement liée au genre vidéoludique FPS. En effet, les jeux conçus grâce à ces moteurs, notamment *Doom* (id Software 1993, Multi), *Quake* (id Software 1996, Multi) ou encore *Unreal* (Epic Games 1998, PC), possèdent effectivement des caractéristiques d'un FPS. Il est donc possible de mettre en relief un lien important entre le genre FPS et la création des premiers moteurs de jeu en 3D temps réel. Un même genre qui stimule également l'innovation technologique, grâce à des exigences techniques élevées (Gregory 2019).

Et, avec l'utilisation de notre modèle de l'abstraction tensive, nous pouvons formuler l'hypothèse que l'uniformisation des multiples technologies implique potentiellement une standardisation des genres vidéoludiques. Cette même standardisation des types de jeux était nécessaire pour développer des moteurs de jeux pertinents et *in fine* les réutiliser autant que possible. Il s'agit donc d'une

limitation technologique, où l'optimisation d'un moteur correspond à un modèle de jeu particulier, qui ne peut néanmoins pas répondre à l'ensemble des exigences d'un autre modèle de jeu.

Naturellement, cela ne signifie pas qu'un moteur est cloisonné à un seul genre particulier, mais plutôt qu'il est *optimisé* pour un genre précis. On peut illustrer cet aspect avec la première version du moteur *Unreal*, utilisé principalement pour les FPS mais également pour des jeux éloignés de ce genre, notamment *Harry Potter and the Sorcerer's Stone* (2001 KnowWonder, PC). Mais nous verrons, dans la prochaine sous-section, que l'industrie vidéoludique va substantiellement évoluer dès la sixième génération de consoles.

7-1-B. DÉMOCRATISATION DES MOTEURS DE JEU

Au regard d'une situation relativement limpide lors de la génération précédente, nous pouvons désormais évoquer les évolutions de l'industrie lors de cette sixième génération de consoles. Celle-ci a débutée dès 1998 avec la *Dreamcast*, puis la *PlayStation 2*, la *Xbox* ainsi que la *GameCube*. Nous avons d'abord observé une démocratisation hypothétique du langage *C++* dans l'industrie vidéoludique, avec pour avantage principal l'utilisation du paradigme orienté objet. Une situation concomitante à celle de la démocratisation éminemment importante des moteurs de jeu commercialisables pour les AAA, phénomène qu'on a illustré à travers les exemples des deux versions de *RenderWare* ou encore du moteur *Unreal Engine 2*. Ces deux exemples illustrent de manière pertinente des précurseurs de notre concept des moteurs 3D universalistes, adaptés à de nombreux genres vidéoludiques malgré des limitations technologiques encore substantielles.

Ce phénomène contrastait avec la situation de l'industrie vidéoludique lors de la cinquième génération. En effet, les nombreux moteurs précédents, notamment le *DOOM Engine*, *Quake Engine* ou *Unreal Engine* étaient d'abord conçus comme étant des moteurs pour le genre FPS, avec un nombre très restreint de jeux pouvant les utiliser pour des genres différents du FPS. Cela à cause notamment de contraintes techniques relativement importantes. En d'autres termes, et dès la sixième génération, les moteurs n'étaient donc plus réellement liés au genre FPS, et semblaient relativement flexibles pour pouvoir être utilisables dans des contextes relativement variés.

Toutefois, les limitations technologiques semblaient encore relativement importantes. On observait initialement des moteurs spécialisés pour certains types de rendus, notamment les *indoor engines* et *outdoor engines*, c'est-à-dire respectivement des moteurs spécialisés en extérieur et des moteurs pour des intérieurs. Plus encore, certains projets nécessitaient la création d'un moteur de jeu intégralement *from scratch*, ce que nous avons observé avec *Metal Gear Solid 3: Snake Eater* (Konami 2004, PS2) ou encore *Animal Crossing: Population Growing!* (Nintendo 2001, GameCube).

Et, au niveau du développement de jeux vidéo amateurs ou indépendants, nous avons observé une démocratisation de nombreux outils, notamment *Irrlicht*,

OGRE, *Crystal Space* ou encore *DarkBasic Pro*. Toutefois, ce phénomène était bien plus restreint sur la console *PlayStation 2*, où le *Linux Kit* était trop limité par rapport au développement professionnel, et le langage *YA-Basic* inadapté pour la conception d'un jeu vidéo en 3D temps réel. En d'autres termes, les multiples concepts sémio-informatiques développés jusqu'à la sixième génération ont permis de mieux appréhender la création de l'interactivité dans les moteurs de jeu. Il est désormais crucial de se tourner, dans la prochaine sous-section, vers la septième génération, dans l'optique de mieux appréhender l'impact de la standardisation des moteurs de jeu sur l'industrie vidéoludique.

7-1-C. RÉUNIFICATION DE L'INDUSTRIE

À l'aune de nos théories portant sur la sixième génération, nous avons d'abord constaté une démocratisation massive des moteurs de jeu. Désormais, nous pouvons synthétiser l'évolution de cette septième génération, ayant commencé dès 2005 via la distribution des consoles *Wii*, *PlayStation 3* et *Xbox 360*. Pour les studios AAA, on a observé une certaine standardisation de l'usage des moteurs internes, réutilisables pour plusieurs projets différents. Mais, surtout, nous avons constaté un phénomène éminemment important lors de cette ère technologique : les prémices d'une réunification de différents types de moteurs pour tendre vers une standardisation totale. En effet, il s'agit d'une convergence entre outils dit professionnels et amateurs, une situation nouvelle dans l'industrie vidéoludique.

Cette même situation a été initiée par la distribution de *XNA* par Microsoft (Sherrod 2008), permettant le développer de jeux vidéo sur PC et *Xbox 360*. Au regard de la *PlayStation 3*, il existait un moteur de jeu disponible pour tous des développeurs indépendants officiels, le *PhyreEngine*. Cela contraste ainsi avec la situation des outils indépendants des générations précédentes, notamment la *Net Yaroze* de la première *PlayStation* ou le *Linux Kit* de la *PlayStation 2*. Des outils qui n'étaient pas réellement conçus pour un développement de jeux professionnels. De surcroît, deux moteurs s'imposeront de manière majoritaire dans l'industrie vidéoludique : *Unity* et *Unreal Development Kit*, la version disponible publiquement du moteur *Unreal Engine 3*. Désormais, nous pouvons exposer les différentes informations de la huitième génération dans la prochaine sous-section.

7-2. STANDARDISATION INDUSTRIELLE PROTÉIFORME

Graduellement, nous avons observé pendant la septième génération une standardisation et réunification importante des moteurs de jeu dans l'industrie vidéoludique. Cela nous permet à présent de synthétiser la huitième génération de consoles, ayant débutée dès 2012 avec la distribution de la *Wii U*, puis de la *PlayStation 4*, de la *Xbox One* ainsi que de la *Switch*. Dans cette génération, on pouvait constater une utilisation plus importante encore des moteurs de jeu commerciaux ou grand public, renforçant la réunification de la génération précédente.

Si de nombreux moteurs disparaissent ou ne parviennent pas à s'imposer dans l'industrie vidéoludique, phénomène qu'on peut illustrer à travers les moteurs *Blender Game Engine* ou encore *Autodesk Stingray*, les statistiques de SteamDB sont relativement limpides : *Unreal Engine* et *Unity* sont désormais des outils majeurs, possédant une popularité éminemment importante, mais également une grande flexibilité.

Les conséquences de cette situation sont nombreuses, mais la plus importante étant de mettre en exergue, au regard de notre problématique, des possibilités très flexibles pour la création des mécanismes interactifs. Cela tend à accentuer la pertinence de notre concept du moteur 3D universaliste, qui semble être une véritable réalité dans l'industrie vidéoludique, et bien au-delà de celle-ci. De surcroît, la métamorphose de l'industrie vidéoludique, notamment avec la fusion des outils pour les professionnels, les indépendants et les amateurs, est donc relativement récente. Nous pouvons à présenter mettre en exergue, dans les trois prochaines sous-sections, nos différents apports scientifiques, ainsi que les difficultés rencontrées.

7-2-A. RÉTROSPECTION SCIENTIFIQUE ET DIFFICULTÉS

Grâce aux analyses informatiques sur notre corpus développemental et vidéoludique, il est désormais possible de mettre en surbrillance les limitations inhérentes de notre approche du *Développement Logiciel Tridimensionnel* ainsi que de notre étude. D'emblée, nous pouvons affirmer que l'ancrage temporel de notre recherche dans les générations de consoles n'était pas aussi pertinent que nous l'avions initialement envisagé. Le développement sur les consoles de salon est, par essence, nettement plus fermé que sur les système d'exploitation *Windows* ou *Mac OS*, et les technologies sont peu connues. C'est pourquoi cette échelle temporelle a engendré des difficultés substantielles, où les informations sur cette partie de l'industrie étaient très fragmentaires, avec un risque de spéculation très important. Naturellement, cela n'occulte aucunement les avantages d'une échelle limpide et synthétique, raison initiale du choix de cette segmentation temporelle pour notre recherche. Il aurait simplement été préférable de concevoir notre propre échelle temporelle technologique, en fonction des résultats de recherches. C'est, hypothétiquement, plus pertinent, voire plus intéressant, que de faire reposer l'ossature de notre recherche sur les générations de consoles. Peut-être une échelle temporelle basée sur les technologies les plus novatrices des moteurs de jeu. Ou encore, de façon plus simplifiée et rétrocompatible avec les générations de consoles, une segmentation axée sur les différentes versions du moteur *Unreal Engine*, qui dans tous les cas est similaire à celle des générations de consoles, mais offrant l'avantage d'être centrée sur les outils de création plutôt que sur l'analyse des consoles elles-mêmes.

Au-delà de cet aspect, et même en exploitant ou créant une autre échelle temporelle, l'une des difficultés majeures résidait également dans l'étude de technologies qui n'ont jamais été publiées, comme les deux premières versions du moteur *Unreal Engine* ou encore les multiples moteurs de jeu propriétaires. Cela a mis en exergue le fait qu'une étude diachronique sur le développement de jeux vidéo, et particulièrement sur consoles, est substantiellement difficile à mener, surtout au regard de la rigueur nécessaire pour une étude scientifique. Elle implique obligatoirement de devoir s'éloigner de l'aspect technologique pour analyser soit un corpus vidéoludique, soit des éléments discursifs qui empêchent

une haute technicité, devenant les seuls moyens de pouvoir répondre à certaines interrogations. Cependant, grâce à cette difficulté cruciale, nous avons réussi à améliorer notre approche sémio-informatique du DLT en formulant le précepte de la comparaison anachronique, permettant de mieux comprendre les technologies ancestrales, en effectuant une comparaison avec celles d'aujourd'hui.

Néanmoins, et c'était un aspect qui a moins été envisagé, nous avons également rencontré de nombreuses difficultés avec des technologies anciennement publiques, mais n'étant plus officiellement disponibles Par exemple, l'analyse de l'outil *Unreal Development Kit*, simplement par l'intermédiaire d'ouvrages techniques anciens ou de la documentation, s'est révélée éminemment difficile. Une difficulté qui subsistait malgré la mise en place de la véridiction technico-discursive, et qui était surtout de l'ordre technique, puisque nous ne pouvions aucunement valider ou invalider certaines descriptions sur ces multiples technologies. C'est pourquoi nous estimons qu'il s'agit d'une méthode qui a légèrement fissuré notre passerelle technologique infrangible, malgré les préceptes précautionneux de notre fondement de l'obsolescence inéluctable. Peut-être aurait-il été préférable d'étudier exclusivement les jeux vidéo commerciaux, provenant de notre corpus vidéoludique, et de comparer ceux-ci seulement à notre corpus développemental direct, en l'occurrence *Unreal Engine 4*. Dans un tel cas, il aurait été plus intéressant de contextualiser les technologies passées seulement en introduction de chaque chapitre, pour ensuite analyser les signes interactifs des multiples jeux. Ce ne sont peut-être pas les caractéristiques technologiques qui sont les plus importantes, mais les résultats pouvant s'exprimer au travers des multiples jeux commerciaux et projets concrets.

De surcroît, et cela pourrait sembler paradoxal avec notre propos précédent, nous avons également constaté une difficulté croissante à obtenir des sources scientifiques, ou pouvant être exploitable scientifiquement, pour les dernières générations de consoles, et particulièrement avec la septième et la huitième génération. Non pas à cause d'une absence de corpus, puisqu'il existe de très nombreux outils de création de jeux, mais également de jeux vidéo. Cette situation est plutôt liée à un manque d'écrits scientifiques dans le cadre d'une recherche doctorale. Cette difficulté substantielle expliquerait pourquoi on peut

observer un déséquilibre quantitatif pour les chapitre portant sur la cinquième et sixième génération, bien plus conséquents que ceux de la septième et huitième génération. Cependant, nous ne pouvons que spéculer sur les raisons pouvant expliquer une difficulté à trouver des sources ou écrits scientifiques sur les dernières générations. En effet, nous pouvons émettre l'hypothèse que les entreprises et acteurs de l'industrie semblent plus enclins à divulguer des informations sur des technologies désuètes et jeux vidéo ancestraux, *in fine* potentiellement inexploitables pour la concurrence, plutôt que les technologies et jeux trop récents. Une solution face à cette problématique aurait été de restreindre notre échelle temporelle aux technologies passées, notamment jusqu'à la sixième génération. Ceci en utilisant le même corpus développemental direct que celui de notre thèse, afin de réaliser un comparatif identique, voire plus détaillé encore, que celui que nous avons réalisé, en évitant *in fine* d'analyser des générations qui semblent trop récentes, où les informations sont relativement fragmentées.

Nous souhaitons enfin aborder un point fondamental, qui n'est pas une difficulté, mais plutôt une spécificité scientifique de notre approche : l'absence délibérée de création de connaissances en informatique. En effet, il est important de ne pas confondre une étude en informatique avec notre recherche, qu'on place officiellement en linguistique, plus précisément en sémiotique appliquée. L'intérêt de notre étude n'est donc nullement d'apporter de nouvelles connaissances en informatique, un objectif qui est déjà parfaitement honoré par l'ensemble des études informatiques ainsi que des multiples industries. En revanche, l'intérêt substantiel de notre étude réside dans la création de théories sémio-informatiques permettant de mieux exploiter les multiples technologies à notre disposition, et non pas de créer de nouvelles technologies. C'est donc la raison principale pour laquelle nos apports scientifiques sont théoriques et conceptuels, plutôt que technologiques, comme en attestent nos deux prochaines sous-sections.

7-2-B. Conception d'une approche scientifique

L'objectif primordial d'une recherche scientifique est de contribuer, d'une manière ou d'une autre, à la progression de la science. Au regard d'un dessein aussi important, et à l'aune des difficultés exposées *supra*, notre but était de pouvoir répondre, autant que possible, à cet impératif de réaliser une recherche contenant des apports scientifiques. D'emblée, nous pouvons naturellement mettre en avant la création d'une nouvelle approche sémio-informatique détaillée de manière exhaustive dans le deuxième chapitre : le Développement Logiciel Tridimensionnel. Celle-ci avait pour but d'analyser et de mieux appréhender le développement de jeux vidéo en 3D temps réel. Surtout, cette innovation permettait de répondre aux limitations plurielles posées par le champ des *Game Studies*, lequel élude les multiples aspects techniques d'une analyse vidéoludique. Or, nous souhaitions fonder une approche qui impliquait rigoureusement de réaliser des analyses relativement techniques, intégrant dans notre corpus un moteur de jeu ou encore différents langages de programmation.

Et, sur le plan scientifique et sémiotique, l'intérêt majeur de cette approche était d'abord de fonder un premier principe immuable, nommé *Passerelle technologique infrangible*. Celui-ci était nécessaire pour créer et consolider un lien entre l'industrie vidéoludique et une théorisation scientifique du développement de jeux vidéo. Il a été constitué de trois fondements distincts : la fortification technologique, l'obsolescence inéluctable et la véridiction technico-discursive. De surcroît, ce premier principe immuable permettait surtout de repenser l'organisation, voire l'applicabilité des *Game Studies* pour l'industrie, en offrant une alternative pour répondre de manière technique aux multiples enjeux et besoins de l'industrie vidéoludique.

Puis nous avons également développé le second principe immuable, intitulé *Générateur de théories vidéoludiques*. Celui-ci a permis de concevoir des théories *ex nihilo* (ou *from scratch*) ou *ex materia* (ou *out of the box*) en incorporant, lorsque c'était possible, des aspects techniques. Celui-ci était également constitué de trois concepts primordiaux : une symbiose sémio-informatique, une interthéoricité logicielle et enfin l'hyperstructure tridimensionnelle. Le premier concept a mis en avant trois types de relations : les oppositions catégorielles, les oppositions graduelles et enfin les relations de complémentarité ou de

combinaison (Courtés 1991 ; Greimas et Courtés 1993 ; Fontanille 2006). Le second concept, nommé interthéoricité logicielle, s'inspire de celui d'interthéoricité (Guillaume 2014 ; Guillaume 2020), mettant en exergue une élasticité des théories et permettant une réappropriation et réadaptation de celles-ci à un autre domaine, ici l'informatique, ou plus précisément le développement de jeux vidéo en 3D temps réel. Enfin, le troisième concept implique de pouvoir *in fine* élaborer la structure ultime sémio-informatique de notre approche du DLT : l'hyperstructure 3D. Cette théorisation scientifique nous semble être la plus importante dans notre recherche, unifiant et consolidant la majorité des théories élaborées. Celle-ci a permis de mettre au jour la création des signes interactifs, autrement dit du *gameplay* ou des mécanismes interactifs, dans les jeux vidéo en 3D temps réel. C'est pourquoi nous allons succinctement synthétiser cette même structure dans la prochaine sous-section.

7-2-C. DÉVELOPPEMENT D'UNE HYPERSTRUCTURE 3D

Au-delà de l'apport d'une approche sémio-informatique novatrice dans notre recherche, on peut mettre en exergue la conception d'une hyperstructure 3D. Celle-ci a permis de réellement renforcer cette passerelle avec l'industrie vidéoludique, que nous avons formalisé sous le concept de la passerelle technologique infrangible. L'objectif principal était bien de pouvoir répondre aux exigences de l'industrie vidéoludique, mais également de pouvoir apporter des théories directement exploitables. Surtout, celle-ci a permis de fournir un cadre théorique novateur pour le développement de jeux vidéo, avec une représentation technologique au sein de théories scientifiques.

Au regard de l'architecture de cette structure sémio-informatique, celle-ci est subdivisée en trois dimensions distinctes : technologique, interactive et structurelle. Ces multiples dimensions sont hiérarchisées, ce qui signifie que la dimension technologique représente la clef de voûte de l'hyperstructure, sur laquelle repose la dimension interactive, puis la dimension structurelle. De surcroît, notre hyperstructure se subdivise également en deux types distincts : interprétative et générative. Le premier type d'hyperstructure permet de pouvoir analyser et interpréter des jeux vidéo commercialisés, autrement dit un corpus vidéoludique. Alors que le second type a plutôt été élaboré pour analyser des moteurs de jeu en eux-mêmes, et établir des théories sur ceux-ci.

Ainsi, la dimension technologique primaire repose sur une nouvelle notion, le moteur 3D universaliste, remplaçant *de facto* celui du moteur de jeu, mais possédant également des concepts distincts. Rajoutons que, dans notre étude, nous analysons exclusivement des moteurs publiquement disponibles, ce que nous pouvons illustrer avec *Unreal Engine* ou encore *Unity*. Et, pour notre définition du moteur 3D universaliste, nous considérons qu'il doit être obligatoirement disponible dans la sphère publique. De surcroît, cette dimension permet surtout de réunir trois caractéristiques fondamentales : l'origine développementale des moteurs, la vision développementale ainsi que l'abstraction tensive.

Puis la dimension interactive permet de mettre en exergue une conceptualisation de l'interactivité plus proche des possibilités technologiques,

cela à travers trois caractéristiques majeures : les signes virtuels interactifs, les sèmes technologiques et les interactivités générationnelles.

Enfin, la dimension structurelle implique de développer la structure d'un monde virtuel dans le processus de développement de jeux vidéo, cela via trois caractéristiques majeures : le triptyque des interactants, les isotopies interactives polymorphiques et enfin la nature structurelle. Dans tous les cas, et grâce à cette hyperstructure 3D, nous avons réussi à répondre à notre problématique initiale et à synthétiser l'ensemble des théories de notre recherche doctorale. De surcroît, nous avons également établi un équilibre majeur entre les analyses purement technologiques et la création des théories sémio-informatiques. Désormais, il est important d'évoquer succinctement les perspectives du développement de logiciels en 3D temps réel, en spéculant quelque peu sur la technologie du futur, et *in fine* les univers virtuels de demain.

7-3. INFINITÉ DES UNIVERS QUANTIQUES

Le développement de jeux vidéo en 3D temps réel représente un océan de potentialités ineffables, où les contraintes technologiques ancestrales périclitent au profit de limitations créatives et humaines. Jadis, les anciennes consoles de jeux contraignaient nos possibilités et proposaient des mondes virtuels aux représentations cubiques, schématiques et simplifiées. De nos jours, nos machines actuelles offrent des possibilités novatrices pour présenter des mondes d'une qualité technique exceptionnelle. Et dans un futur proche, elles s'affranchiront des barrières que nous connaissons pour n'avoir qu'une seule et unique limite : la créativité. C'est pourquoi, eu égard à la création de notre approche sémio-informatique novatrice, le Développement Logiciel Tridimensionnel, et de sa structure sémio-informatique, l'hyperstructure des logiciels 3D, il reste un nombre gargantuesque de chemins à explorer.

Plus encore, nous pourrions plutôt évoquer une infinité de perspectives inexplorées. Certes, on pourrait d'abord mentionner les différents aspects directement occultés par cette recherche : l'interactivité dans les jeux en ligne, l'impact des périphériques comme le casque de réalité virtuelle ou encore le *gameplay* des jeux sur différentes machines comme les smartphones ; autant de prismes capables d'augmenter substantiellement la compréhension des mécanismes interactifs et *in fine* d'élaborer une hyperstructure 3D toujours plus exhaustive. Cependant, nous considérons que l'informatique est à l'orée d'une véritable révolution qui bouleversera les jeux vidéo en 3D temps réel et, d'une manière plus significative encore, notre société contemporaine. Un événement quasi-transcendantal, susceptible de reléguer instantanément toutes nos machines actuelles dans une ère préhistorique : la technologie quantique.

Hypothétiquement, un ordinateur quantique pourrait disposer d'une puissance colossale et titanesque, et, comme le précise Gill *et al.*, « Google prétend avoir atteint la suprématie quantique (...) [et] créé une machine capable d'accomplir en 4 minutes ce qui demanderait au super-ordinateur le plus performant de la planète 10 000 ans pour réaliser la même tâche » (Gill *et al.* 2024, ch. 4.3.2

TDR)[575], une machine si puissante qu'elle pourrait générer des univers virtuels dans lesquels des concepts aujourd'hui avant-gardistes seraient dépassés. Naturellement, cette technologie futuriste pourrait offrir des possibilités qui sont encore de l'ordre de la spéculation.

Néanmoins, la démocratisation des technologies quantiques dans l'industrie vidéoludique exigera de s'émanciper des limitations imposées par certains concepts théoriques sur le *gameplay*. Notre théorie de la vision développementale du monde virtuel implique de repenser la création des mécanismes interactifs pour chaque nouvelle génération, autrement dit lors de chaque nouveau palier technologique. Cela a permis à notre modèle sémio-informatique de l'hyperstructure 3D de développer une architecture respectant trois dimensions fondamentales mais ancrées dans une ère ou une génération de consoles donnée. De surcroît, nous avons eu pour objectif de respecter les multiples principes de la programmation, permettant d'intégrer des concepts informatiques dans nos théories. On peut également estimer que nos concepts possèdent des fondements théoriques suffisamment solides et flexibles pour répondre et s'adapter à des enjeux nouveaux de la prochaine ère de l'informatique.

Il est désormais temps de clôturer définitivement cette thèse doctorale, en mettant en surbrillance les possibilités infinies dans les jeux vidéo en 3D temps réel. Un domaine qui, pour moi, exerce une fascination indicible et éternelle, dans lequel nous pouvons offrir aux joueurs la possibilité de voyager dans des univers virtuels aux temps immémoriaux. De surcroît, notre industrie vidéoludique est capable de développer des mondes virtuels les plus immersifs possibles, d'une richesse potentiellement illimitée. Et nous, les développeurs de jeux vidéo 3D, avons tous un rôle à jouer pour construire ce futur que vous, chercheurs, pourrez ensuite analyser puis théoriser.

575. « Google claimed to have achieved quantum supremacy (...) [and] created a machine that can perform in 4 min whatever would require the most capable supercomputer on the planet 10,000 years to achieve » (Gill *et al.* 2024, ch. 4.3.2).

Enfin, dans un futur proche, peut-être que le rêve chimérique, voire utopique, de développer un univers virtuel plus vaste et riche que la Voie lactée deviendra, un jour, réalité. Un tel univers offrirait une richesse si gargantuesque qu'une mémoire humaine, même eidétique, ne pourrait sauvegarder. Un même univers virtuel qui pourrait simultanément réunir tous les joueurs de la planète dans un seul et même espace, pour une interactivité protéiforme, multimédia et multifacette. Il s'agit de l'aspiration ultime des moteurs 3D universalistes : développer un jeu possédant une interactivité plurielle dans un univers virtuel interconnecté infini...

RÉFÉRENCES

BIBLIOGRAPHIE

Aarseth, Espen. 2001. « Computer Game Studies, Year One ». *Game Studies* 1 (1). https://www.gamestudies.org/0101/editorial.html. [Consulté le 01/03/2024]

Aarseth, Espen. 2015. « Meta-Game Studies ». *Game Studies* 15 (1). http://gamestudies.org/1501/articles/editorial. [Consulté le 01/03/2024]

Aarseth, Espen. 2017. « Just Games ». *Game Studies* 17 (1). http://gamestudies.org/1701/articles/justgames. [Consulté le 01/03/2024]

Académie française. (1992) 2024. « Action ». In *Dictionnaire de l'Académie française*, 9e éd. https://www.dictionnaire-academie.fr/article/A9A0471. [Consulté le 01/03/2024]

Académie française. (1992) 2024. « Amateur ». In *Dictionnaire de l'Académie française*, 9e éd. https://www.dictionnaire-academie.fr/article/A9A1388. [Consulté le 01/03/2024]

Akchelov, Evgenii, et Ekaterina Galanina. 2016. « Virtual World of Video Games ». In *IEEE 8th International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES)*, 1-4. https://doi.org/10.1109/VS-GAMES.2016.7590379. [Consulté le 01/03/2024]

Akenine-Möller, Tomas, Eric Haines, Naty Hoffman, Angelo Pesce, Michał Iwanicki, et Sébastien Hillaire. (1999) 2018. *Real-Time Rendering*. 4e éd. Florida : CRC Press Online. 1178.

Altice, Nathan. 2015. *I am Error: The Nintendo Family Computer / Entertainment System Platform*. Cambridge : MIT Press. 426.

Amato, Étienne Armand. 2014. « Le jeu vidéo, premier média cybernétique de masse ». *Hermès, La Revue* 70 (3) : 163-65. https://doi.org/10.3917/herm.070.0163. [Consulté le 01/03/2024]

American National Standards Institute. 1990. *American National Standard for Programming Languages – C*. ANSI/ISO 9899-1990. New York : American National Standards Institute. 220.

Amos, Evan. 2021. *Les Consoles de jeux vidéo : Ces consoles qui ont fait l'histoire du jeu vidéo*. Avon : Omake books. 251.

Anderson, Craig A., et Brad J. Bushman. 2001. « Effects of Violent Video Games on Aggressive Behavior, Aggressive Cognition, Aggressive Affect, Physiological Arousal, and Prosocial Behavior: A Meta-Analytic Review of the

Scientific Literature ». *Psychological Science* 12 (5): 353-59. https://doi.org/10.1111/1467-9280.00366. [Consulté le 01/03/2024]

Arnaud, Rémi. 2011. « The Game Asset Pipeline ». In *Game Engine Gems 1, édité par Eric Lengyel*, 15-52. Ontario : Jones & Bartlett Publishers.

Arsenault, Dominic. 2011. « Des typologies mécaniques à l'expérience esthétique. Fonctions et mutations du genre dans le jeu vidéo ». Thèse doctorale, Université de Montréal. 352.

Aurnague, Michel. 2004. *Les structures de l'espace linguistique. Regards croisés sur quelques constructions spatiales du basque et du francais*. Louvain : Peeters Leuven. 272.

Bacone, Victor Kuller. 2012. *Blender Game Engine: Beginner's Guide*. Birmingham: Packt Publishing. 206.

Badir, Sémir. 2010. « Sémiotique de la connaissance ». *Signata* 2010 (1) : 239-53. https://doi.org/10.4000/signata.305. [Consulté le 01/03/2024]

Baer, Ralph Henry. 2005. *Videogames: In the Beginning*. Springfield : Rolenta Press. 280.

Barthes, Roland. 1964a. « Éléments de sémiologie ». Communications 4 : 91-135.

Barthes, Roland. 1964b. « Présentation ». Communications 4: 1-3.

Barton, Matt. 2019. *Vintage Games 2.0: An insider look at the most influential games of all time*. Florida : CRC Press. 357.

Batu, Bengu. 2012. « An Overview of the Field of Semiotics ». *Procedia - Social and Behavioral Sciences* 51 : 464-469. https://doi.org/10.1016/j.sbspro.2012.08.190. [Consulté le 01/03/2024]

Berthelot-Guiet, Karine. 2004. « Instrumentalisations de la sémiotique ». *Études de communication* 2004 (27) : 1-9. https://doi.org/10.4000/edc.148. [Consulté le 01/03/2024]

Berthelot-Guiet, Karine, et Jean-Jacques Boutaud. 2015. « Sémiotique, mode d'emploi : Avant-propos ». In *Sémiotique, mode d'emploi*, sous la direction de Karine Berthelot-Guiet et Jean-Jacques Boutaud. Lormont : Le Bord de l'Eau. 5-14. https://shs.hal.science/halshs-01139345. [Consulté le 01/03/2024]

Bezchotnikova, Svitlana, et Alina Bezchotnikova. 2018. « Game Mechanics as Videogame Genre Identifier ». *Global media journal* 16 (30): 1-8.

Bilińska-Reformat, Katarzyna, Anna Dewalska-Opitek, et Magdalena Hofman-Kohlmeyer. 2020. « To Mod or Not to Mod—An Empirical Study on Game

Modding as Customer Value Co-Creation ». *Sustainability* 12 (21) : 1-16. https://doi.org/10.3390/su12219014. [Consulté le 01/03/2024]

Blackwell, Allan, et Kerry Rodden. 2003. « Preface ». In *Sketchpad: A man-machine graphical communication system*, sous la direction de Ivan Edward Sutherland. Cambridge : University of Cambridge. 3-6.

Blanchet, Alexis. 2010. *Des pixels à Hollywood : Cinéma et jeu vidéo, une histoire économique et culturelle*. Châtillon : Pix'n Love. 450.

Blum, Richard. 2005. *Professional Assembly Language*. Indianapolis : Wiley Publishing. 546.

Bonenfant, Maude. 2010. « Sens, fonction et appropriation du jeu : l'exemple de World of Warcraft ». Thèse doctorale, Université du Québec à Montréal. 474.

Bossom, Andy, et Ben Dunning. 2015. *Video Games: An Introduction to the Industry*. New York: Bloomsbury Publishing. 208.

Bouquet, Simon. 2004. « Après un siècle, les manuscrits de Saussure reviennent bouleverser la linguistique ». *Texto !* juin 2005. http://www.revue-texto.net/1996-2007/Saussure/Sur_Saussure/Bouquet_Apres.html. [Consulté le 01/03/2024]

Bouquet, Simon. 2012. « Principes d'une linguistique de l'interprétation : une épistémologie néosaussurienne ». *Langages* 185 (1) : 21-33. https://doi.org/10.3917/lang.185.0021. [Consulté le 01/03/2024]

Boutellier, Roman, Oliver Gassmann, et Maximilian von Zedtwitz. (1999) 2008. « Canon: R&D –Driver for Continuous Growth and Diversification ». In *Managing Global Innovation: Uncovering the Secrets of Future Competitiveness*, 3e éd., 379-395. Berlin: Springer-Verlag.

Bouveresse, Renée. 1978. *Karl Popper ou le rationalisme critique*. Paris : Librairie philosophique J. Vrin.

Brooks. 1987. « No Silver Bullet Essence and Accidents of Software Engineering ». *Computer* 20 (4): 10-19. https://doi.org/10.1109/MC.1987.1663532. [Consulté le 01/03/2024]

Bross, Alexis, et Loup Lassinat-Foubert. 2018. *Générations Mario : C'est l'histoire d'un plombier....* Toulouse : Third Editions. 191.

Bryan, Kevin. 2005. *PSX: The guide to the Sony PlayStation*. Digital Press. 139.

Bubsy, Jason, Zak Parrish, et Jeff Wilson. 2009. *Mastering Unreal Technology, Volume I: Introduction to Level Design with Unreal Engine 3*. Indianapolis : Sams Publishing. 850.

Bubsy, Jason, Zak Parrish, et Jeff Wilson. 2010. *Mastering Unreal Technology, Volume II: Advanced Level Design Concepts with Unreal Engine 3*. Indianapolis : Sams Publishing. 1168.

Burns, Matthew. 2010. « Unity Technologies: Unity Pro 3 ». *Game Developer Magazine* August 2010 : 34-36.

Byl, Penny de. 2014. *Holistic Mobile Game Development With Unity*. New York : Routledge. 576.

Calzolari, Nicoletta. 2009. « Computational Linguistics ». In *Linguistics, édité par Vesna Muhvic-Dimanovski et Lelija Socanac*, 328-345. Oxford : EOLSS Publishers.

Carlo, Catherine. 2015. « Pluri-inter-transdisciplinarité des recherches impliquées dans la description de la langue en usage : Comment penser la porosité des champs de recherche ? » *Recherches en didactique des langues et des cultures* 12-3 : 1-9. https://doi.org/10.4000/rdlc.959. [Consulté le 01/03/2024]

Cass, Stephen. 2023. « The Top Programming Languages 2023 Python and SQL are on top, but old languages shouldn't be forgotten ». IEEE Spectrum. 2023. https://spectrum.ieee.org/the-top-programming-languages-2023. [Consulté le 01/03/2024]

Chandler, Daniel. (2001) 2022. *Semiotics: The Basics*. 4e éd. New York : Routledge.

Chandler, Heather Maxwell, et Rafael Chandler. 2011. *Fundamentals of Game Development*. Ontario: Jones & Bartlett Learning. 379.

Charaudeau, Patrick. 2010. « Pour une interdisciplinarité « focalisée » dans les sciences humaines et sociales ». *Questions de communication* 2010 (17) : 195-222. https://doi.org/10.4000/questionsdecommunication.385. [Consulté le 01/03/2024]

Chaumartin, François-Régis, et Pirmin Lemberger. 2020. *Le traitement automatique des Langues : Comprendre les textes grâce à l'intelligence artificielle*. Malakoff : Dunod. 336.

Chauviré, Christiane. 1995. *Peirce et la signification : Introduction à la logique du vague*. Paris : Presses Universitaires de France. 290.

Chen, Jiahao. 2018. « Linguistic Relativity and Programming Languages ». In *Proceedings of the 2016 Joint Statistical Meetings*. Chicago, USA. https://arxiv.org/abs/1808.03916. [Consulté le 01/03/2024]

Christopoulou, Eleftheria, et Stelios Xinogalos. 2017. « Overview and Comparative Analysis of Game Engines for Desktop and Mobile Devices ». *International Journal of Serious Games* 4 (4) : 21-36. https://doi.org/10.17083/ijsg.v4i4.194. [Consulté le 01/03/2024]

Clark, Dan. (2011) 2013. *Beginning C# Object-Oriented Programming: Learn to design and develop modern applications in C#*. 2e éd. New York : Apress. 360.

Clarke, Rachel Ivy, Jin Ha Lee, et Neils Clark. 2015. « Why Video Game Genres Fail: A Classificatory Analysis ». *Games and Culture* 12 (5) : 1-22. https://doi.org/10.1177/1555412015591900. [Consulté le 01/03/2024]

Coavoux, Samuel, Manuel Boutet, et Vinciane Zabban. 2016. « What We Know About Games: A Scientometric Approach to Game Studies in the 2000s ». *Games and Culture* 12 (6) : 563-84. https://doi.org/10.1177/1555412016676661. [Consulté le 01/03/2024]

Cordone, Rachel. 2019. *Unreal Engine 4 Game Development Quick Start Guide*. Birmingham : Packt Publishing. 185.

Couégnas, Nicolas. 2022. « L'épistémologie critique de la sémiotique ». *Signata* 2022 (13). https://doi.org/10.4000/signata.3859. [Consulté le 01/03/2024]

Courcier, Nicolas, Mehdi El Kanafi, et Raphaël Lucas. 2019. *BioShock: From Rapture to Columbia*. Toulouse: Third Editions. 192.

Courtés, Joseph. 1991. Analyse sémiotique du discours : de l'énoncé à l'énonciation. Paris : Hachette supérieur. 302.

Courtés, Joseph. 2009. « L'énonciation, point de départ de l'analyse sémiotique ». *Anadiss 7* (1) : 15-33.

Crawford, Chris. 1984. *The Art Of Computer Game Design: Reflections Of A Master Game Designer*. California: Osborone/ McGraw-Hill. 113.

Crooks II, Claytone E. 2003. The gmax Handbook. Hingham : Charles River Media. 357.

Daim, Tugrul, Jay Justice, Liliya Hogaboam, Saku J. Mäkinen, et Ozgur Dedehayir. 2014. « Identifying and forecasting the reverse salient in video game consoles: A performance gap ratio comparative analysis ». *Technological Forecasting*

and Social Change 82 : 177-189. https://doi.org/10.1016/j.techfore.2013.06.007. [Consulté le 01/03/2024]

Dale, Gillian, et C. Shawn Green. 2017. « The Changing Face of Video Games and Video Gamers: Future Directions in the Scientific Study of Video Game Play and Cognitive Performance ». *Journal of Cognitive Enhancement* 1 (3) : 280-294. https://doi.org/10.1007/s41465-017-0015-6. [Consulté le 01/03/2024]

Darnell, Peter A., et Philip E. Margolis. 1996. *C A Software Engineering Approach*. 3e éd. Massachusetts : Springer-Verlag. 497.

Dawson, Michael. (2004) 2014. *Beginning C++ Through Game Programming*. 4e éd. Kentucky : Cengage Learning PTR. 447.

DeLoura, Mark. 1999. « Nintendo 64 Programming ». *Game Developer Magazine* November 1999 : 30-41.

DeMaria, Rusel. 2019a. *Game of X V.1 Xbox*. Florida: CRC Press. 245.

DeMaria, Rusel. (2002) 2019b. *High Score! Expanded: The illustrated history of electronic games*. 3e éd. Florida: CRC Press. 439.

Depecker, Loïc. 2012. « Les manuscrits de Saussure : une révolution philologique ». *Langages* 185 (1) : 3-6. https://doi.org/10.3917/lang.185.0003. [Consulté le 01/03/2024]

Deterding, Sebastian. 2016. « The Pyrrhic Victory of Game Studies: Assessing the Past, Present, and Future of Interdisciplinary Game Research ». *Games and Culture* 12 (6) : 1-25. https://doi.org/10.1177/1555412016665067. [Consulté le 01/03/2024]

Di Filippo, Laurent. 2012. « Les notions de personnage-joueur et Roleplay pour l'étude de l'identité dans les MMORPG ». *Revue ¿ Interrogations ?*, Identité fictive et fictionnalisation de l'identité, 15. https://www.revue-interrogations.org/Les-notions-de-personnage-joueur. [Consulté le 01/03/2024]

Di Filippo, Laurent. 2014. « Contextualiser les théories du jeu de Johan Huizinga et Roger Caillois ». *Questions de communication* 25 : 281-308. https://doi.org/10.4000/questionsdecommunication.9044. [Consulté le 01/03/2024]

Diefendorff, Keith. 1999. « Sony's Emotionally Charged Chip: Killer Floating-Point "Emotion Engine" To Power PlayStation 2000 ». *Microcrocessor Report* 13 (5) : 1-7.

Dillon, Roberto. 2011. *The Golden Age of Video Games: The Birth of a Multi-Billion Dollar Industry*. Florida: CRC Press. 172.

Dixit, J. B. (2005) 2011. *Programming in C*. 3e éd. New Delhi : Firewal Media. 619.

Djaouti, Damien. 2019. *Préhistoire du jeu vidéo*. Houdan : Pix'n Love. 127.

Dogruel, Leyla, et Sven Joeckel. 2013. « Video game rating systems in the US and Europe: Comparing their outcomes ». *International Communication Gazette* 75 (7) : 672-692. https://doi.org/10.1177/1748048513482539. [Consulté le 01/03/2024]

Doran, John P., William Sherif, et Stephen Whittle. 2019. *Unreal Engine 4.x Scripting with C++ Cookbook: Develop quality game components and solve scripting problems with the power of C++ and UE4*. Birmingham: Packt Publishing. 673.

Dosse, François. 1991. *Histoire du structuralisme. Tome I : Le champ du signe. 1945-1966*. Paris : La Découverte.

Dozo, Björn-Olav. 2019. « Pour une histoire polyphonique du jeu vidéo ». In *Culture vidéoludique!*, sous la direction de Liège Game Lab. Liège: Presses universitaires de Liège. 21-35.

Drummond, Aaron, James D. Sauer, et Christopher J. Ferguson. 2020. « Do longitudinal studies support long-term relationships between aggressive game play and youth aggressive behaviour? A meta-analytic examination ». *Royal Society Open Science* 7 (7). https://doi.org/10.1098/rsos.200373. [Consulté le 01/03/2024]

Dunne, Alex. 1995. « 3D Graphics Goliaths Square Off ». *Game Developer Magazine* June/ July 1995 : 11-14.

Eberly, David H. 2005. *3D Game Engine Architecture: Engineering Real-Time Applications with Wild Magic*. San Francisco: Elsevier. 736.

Eco, Umberto. (1977) 2015. *How to Write a Thesis*. Cambridge : MIT Press. 230.

Egenfeldt-Nielsen, Simon, Jonas Heide Smith, et Susana Pajares Tosca. 2015. *Understanding Video Games: The Essential Introduction*. 3e éd. New York : Routledge. 286. https://doi.org/10.4324/9781315725161. [Consulté le 01/03/2024]

Elverdam, Christian, et Espen Aarseth. 2007. « Game Classification and Game Design: Construction Through Critical Analysis ». *Games and Culture* 2 (1) : 3-22. https://doi.org/10.1177/1555412006286892. [Consulté le 01/03/2024]

Engström, Henrik. 2019. « GDC vs. DiGRA: Gaps in Game Production Research ». In *Proceedings of the 2019 DiGRA International Conference: Game, Play and the Emerging Ludo-Mix*.

Eraly, Alain. 2011. « Quelle sémiotique pour quelle théorie sociale ? » *Signata* 2011 (2) : 167-194. https://doi.org/10.4000/signata.655. [Consulté le 01/03/2024]

Ernkvist, Mirko. 2008. « Down Many Times, but Still Playing the Game: Creative Destruction and Industry Crashes in the Early Video Game Industry 1971-1986 ». In *History of Insolvency and Bankruptcy: from an International Perspective,* édité par Karl Gratzer et Dieter Stiefel. Huddinge: Södertörns högskola. 161-191.

Fackroth, Christian, Rico Kubitza, Lars Stockmann, Jens Holze, Roland Winkler, Niklas Röber, et Maic Masuch. 2005. « PS2 game development under Linux ». In *Computer science and magic 2005*.

Feldman, James M., et Edward W. Czeck. 2006. « Programming: Assembly Language ». In *Computers, Software Engineering, and Digital Devices,* édité par Richard C. Dorf. Florida : CRC Press.

Ferguson, Christopher John. 2015. « Do Angry Birds Make for Angry Children? A Meta-Analysis of Video Game Influences on Children's and Adolescents' Aggression, Mental Health, Prosocial Behavior, and Academic Performance ». *Perspectives on Psychological Science* 10 (5) : 646-666. https://doi.org/10.1177/1745691615592234. [Consulté le 01/03/2024]

Fils-Aimé, Reggie. 2022. *Disrupting the Game: From the Bronx to the top of Nintendo*. California: HarperCollins Leadership. 224.

Forestier, Georges. 2017. *Introduction à l'analyse des textes classiques*. 5e éd. Paris : Armand Colin. 160.

Fontana, Richard. 2022. « Cloud Native Development, Containers, and Open Source Licensing ». In *Open Source Law, Policy and Practice, édité par Amanda Brock*. Oxford : Oxford University Press.

Fontanille, Jacques. 2006. *The Semiotics of Discourse*. New York : Peter Lang Publishing. 210.

Fontanille, Jacques. 2010. « L'analyse des pratiques : le cours du sens ». *Protée* 38 (2) : 1-14.

Fontanille, Jacques, et Alessandro Zinna. 2019. « Le dialogue entre la sémiotique structurale et les sciences. Hommage à Algirdas Julien Greimas ». *Langages* 213 (1) : 5-16. https://doi.org/10.3917/lang.213.0005. [Consulté le 01/03/2024]

Galanina, Ekaterina, Evgenii Akchelov, et Elena Sakharova. 2020. « Ontology of Video Game Virtual Worlds ». *Vision 2020: Innovation Management, Development Sustainability, and Competitive Economic Growth* : 33-43.

Genvo, Sébastien. 2006. « Le Game Design de jeux vidéo : Approche communicationnelle et interculturelle ». Thèse doctorale, Université Paul Verlaine – Metz. 405.

Sarfati, Georges-Elia. 2020. *Linguistique : Initiation aux grandes théories*. 2e éd. Malakoff : Armand Colin. 286.

Gill, Tripat. 2008. « Convergent Products: What Functionalities Add More Value to the Base? » *Journal of Marketing* 72 (2) : 46-62. https://doi.org/10.1509/jmkg.72.2.46. [Consulté le 01/03/2024]

Gill, Sukhpal Singh, Huaming Wu, Panos Patros, Carlo Ottaviani, Priyansh Arora, Victor Casamayor Pujol, David Haunschild, et al. 2024. « Modern computing: Vision and challenges ». *Telematics and Informatics Reports* 13 (mars): 100116. https://doi.org/10.1016/j.teler.2024.100116. [Consulté le 01/03/2024]

Greenberg, Raz. 2021. « The Animation of Gamers and the Gamers as Animators in Sierra On-Line's Adventure Games ». *Animation* 16 (1-2) : 83-95. https://doi.org/10.1177/17468477211025665. [Consulté le 01/03/2024]

Graham, Raymond. 2018. « A Lost Art ». In *Behind the Black Box: Sessions with Game Engine Professionals,* édité par Caleb Biasco, Jared Ettinger, Jacob Wilson, Chaojie Zhu, et Yidi Zhu. Pittsburgh : ETC Press. 98-108.

Gregoire, Marc. (2011) 2021. *Professional C++*. 5e éd. Indianapolis : John Wiley & Sons. 1256.

Gregory, Jason. (2009) 2019. *Game Engine Architecture*. 3e éd. Florida : CRC Press. 1200.

Greimas, Algirdas-Julien, et Joseph Courtés. (1979) 1993. *Sémiotique : Dictionnaire raisonné de la théorie du langage*. Paris : Hachette Livre. 454.

Guillaume, Astrid. 2014. « L'interthéoricité : sémiotique de la transférogenèse. Plasticité, élasticité, hybridité des théories ». In *PLASTIR, Plasticités, Sciences et Arts (37) 12 :* 17-53. https://plasticites-sciences-arts.org/PLASTIR/Guillaume%20P37.pdf. [Consulté le 01/03/2024]

Guillaume, Astrid. 2020. « Sémiotique des cultures et hybridités théoriques : pour un renouvellement de la pensée ». In *Faire sens, faire science, édité par Astrid Guillaume et Lia Kurts Wöste*, 121-144. London : ISTE Editions.

Guo, Philip J. 2018. « Non-Native English Speakers Learning Computer Programming: Barriers, Desires, and Design Opportunities ». In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1-14. CHI '18. New York : Association for Computing Machinery. https://doi.org/10.1145/3173574.3173970. [Consulté le 01/03/2024]

Haigh, Thomas, et Paul E. Ceruzzi. 2021. *A New History of Modern Computing*. Cambridge : MIT Press. 530.

Hall, Brian R., et Kevin J. Slonka. 2018. *Assembly Programming and Computer Architecture*. Danvers : Prospect Press.

Hanania, Cécile. 2010. *Roland Barthes et l'étymologie*. Bruxelles : Peter Lang Publishing.

Hansen, Dustin. 2016. *Game On! Video Game History from Pong and Pac-Man to Mario, Minecraft, and More*. New York: Feiwel & Friends. 368.

Harbour, Jonathan, et Joshua Smith. 2003. *Beginner's Guide to DarkBASIC Game Programming*. Cincinnati : Premier Press. 711.

Hawkins, Kevin, Dave Astle, et André LaMothe. 2001. *OpenGL Game Programming*. California: Prima Publishing. 777.

Hébert, Louis. 2005. « Le schéma tensif : synthèse et propositions ». *Tangence* 79 : 111-139. https://doi.org/10.7202/012854ar. [Consulté le 01/03/2024]

Hébert, Louis. 2020. *An introduction to applied semiotics: Tools for Text and Image Analysis*. New York: Routledge. 290.

Hemmerling, Marco. 2020. « 3D Modelling ». In *Atlas of Digital Architecture: Terminology, Concepts, Methods, Tools, Examples, Phenomena*, édité par Ludger Hovestadt, Urs Hirschberg et Oliver Frit, 57-93. Basel: Birkhäuser.

Hénault, Anne. 2008. « Image et texte au regard de la sémiotique ». *Le français aujourd'hui* 161 (2) : 11-20. https://doi.org/10.3917/lfa.161.0011. [Consulté le 01/03/2024]

Hermès. 2012. « Typologie des jeux vidéo ». *Hermès, La Revue* 62 (1) : 15-16. https://doi.org/10.4267/2042/48270. [Consulté le 01/03/2024]

Hickey, Patrick Jr. 2020. *Minds behind adventure Games: Interviews with Cult and Classic Video Game Developers*. North Carolina : McFarland. 226.

Hickey, Patrick Jr. 2021. *The Minds Behind Shooter Games: Interviews with Cult and Classic Video Game Developers*. North Carolina : McFarland. 263.

Hickey, Patrick Jr. 2022. *The Minds Behind PlayStation Games: Interviews with Creators and Developers*. North Carolina : McFarland. 291.

Hodent, Celia. 2021. *The psychology of video games*. New York : Routledge. 104.

Hoffin, Kevin, et Elaine DeVos. 2020. « A Chronology of Video Game Deviance ». In *Video Games, Crime and Next-Gen Deviance*, édité par Craig Kelly, Adam Lynes, et Kevin Hoffin, 49-72. Bingley: Emerald Publishing Limited.

Horowitz, Ken. 2016. *Playing at the next level : a history of American Sega Games*. North Carolina : McFarland.

Horton, Ivor. 2013. *Beginning C: Takes you step-by-step from novice to C programmer*. 5e éd. New York : Springer Science + Business Media.

Horton, Ivor, et Peter Van Weert. (1998) 2020. *Beginning C++20: From Novice to Professional*. 6e éd. New York: Apress. 825.

Hugg, Steven. 2018. *Designing Video Game Hardware in Verilog: An 8bitworkshop Book*. Steven Hugg.

Hugg, Steven. 2019. *Making Games for the NES: An 8bitworkshop Book*. Steven Hugg.

Hunt, Andy, et Dave Thomas. 2002. « Software Archaeology ». *IEEE Software.* 19 (2) : 20-22. https://doi.org/10.1109/52.991327. [Consulté le 01/03/2024]

Hyde, Randall. (2004) 2020a. *Write Great Code, Volume 1: Understanding the machine*. 2e éd. San Francisco : No Starch Press. 452.

Hyde, Randall. (2006) 2020b. *Write Great Code, Volume 2: Thinking Low Level, writing high level*. 2e éd. San Francisco : No Starch Press. 629.

Ichbiah, Daniel. 2014. *La Saga des Jeux Vidéo*. 6e éd. Houdan: Pix'n Love. 610.

Imagine Media. 1996. « The New Psy-Q PlayStation Plug-in ». *Next Generation: Leading edge computer and video games* 15 (March 1996) : 80-80.

Järvinen, Aki. 2002. « Halo and the Anatomy of the FPS ». *Game Studies* 2 (1). http://www.gamestudies.org/0102/jarvinen/. [Consulté le 01/03/2024]

Justice, Matthew. 2021. *How Computers really work: A hands-on guide to the inner workings of the machine*. San Francisco : No Starch Press. 366.

Juul, Jesper. 2000. « What computer games can and can't do ». In *Digital Arts and Culture Conference*. Norvège. http://www.jesperjuul.net/text/wcgcacd.html. [Consulté le 01/03/2024]

Kaleniuk, Oleksandr. 2023. *Geometry for Programmers*. New York : Manning Publications. 416.

Kent, Steven L. 2001. *The Ultimate History of Video Games: From Pong to Pokémon and Beyond- The Story Behind the Craze That Touched Our Lives and Changed the World*. New York : Three Rivers Press. 608.

Kent, Steven L. 2021. *The Ultimate History of Video Games: Volume 2 – Nintendo, Sony, Microsoft, and the Billion-Dollar Battle to Shape Modern Gaming*. New York: Penguin Random House.

Kernighan, Brian W., et Dennis M. Ritchie. 1988. *The C Programming Language*. 2e éd. New York : Pearson. 272.

Kleinpeter, Édouard. 2013. « Taxinomie critique de l'interdisciplinarité ». *Hermès, La Revue* 67 (3) : 123-29. https://doi.org/10.4267/2042/51898.

Klett, Steve. 2004. Maximum PC : The Ultimate 3D Gaming Tech Guide. Future Network USA : San Francisco. 104.

Klinkenberg, Jean-Marie. (1996) 2000. *Précis de sémiotique générale*. Paris : Éditions Points. 488.

Klinkenberg, Jean-Marie. 2010. « La sémiotique visuelle : grands paradigmes et tendances lourdes ». *Signata* 2010 (1) : 91-109. https://doi.org/10.4000/signata.287. [Consulté le 01/03/2024]

Kovach, Peter J. 1998. *The Awesome Power of Direct3D/ DirectX*. New York : Manning Publications.

Kutaragi, Ken. 2018. « PlayStation: Early Efforts in the Creation and Development of the Digital Entertainment Market ». *Ritsumeikan University Game Research Center/Replaying Japan* Inaugural preparatory issue: 53-59.

Lammes, Sybille. 2007. « Approaching game-studies: towards a reflexive methodology of games as situated cultures ». In *DiGRA '07 - Proceedings of the 2007 DiGRA International Conference: Situated Play*. The University of

Tokyo. http://www.digra.org/wp-content/uploads/digital-library/07311.28016.pdf. [Consulté le 01/03/2024]

Landay, Lori. 2014. « Interactivity ». In *The Routledge Companion to Video Game Studies*, édité par Mark J. P. Wolf et Bernard Perron, 173-184. New York : Routledge.

Landowski, Eric. 2004. *Passions sans nom : Essais de socio-sémiotique III*. Paris : Presses Universitaires de France. 316.

Landowski, Eric. 2022. « Les métamorphoses de la vérité, entre sens et interaction ». *Revista Acta Semiotica* 2 (3) : 256-75. https://doi.org/10.23925/2763-700X.2022n3.58417. [Consulté le 01/03/2024]

Le Bihan, Baptiste. 2016. « Les propriétés du vide et de l'espace-temps ». *Philosophiques* 43 (1) : 49-66. https://doi.org/10.7202/1036467ar. [Consulté le 01/03/2024]

Le Diberder, Alain, et Frédéric Le Diberder. 1998. *L'univers des Jeux Vidéo*. Paris : La Découverte. 228.

Leconte, Gauvain. 2019. « Théorie de la définition et jeu vidéo : règles et lois dans l'expérience vidéolududique ». *Sciences du jeu* 2019 (11). https://doi.org/10.4000/sdj.1662. [Consulté le 01/03/2024]

Ledin, Jim. (2020) 2022. *Modern Computer Architecture and Organization: Learn x86, ARM, and RISC-V architectures and the design of smartphones, PCs, and cloud servers*. 2e éd. Birmingham : Packt Publishing. 628.

Lee, Jin Ha, Natascha Karlova, Rachel Ivy Clarke, Katherine Thornton, et Andrew Perti. 2014. « Facet Analysis of Video Game Genres ». In *iConference 2014 Proceedings*, 125-139. iSchools. https://doi.org/10.9776/14057. [Consulté le 01/03/2024]

Lee, Joanna. 2016. *Learning Unreal Engine Game Development: A step-by-step guide that paves the way for developing fantastic games with Unreal Engine 4*. Birmingham: Packt Publishing. 249.

Lendino, Jamie. 2020. *Attract Mode: The Rise and Fall of Coin-Op Arcade Games*. Audubon : Steel Gear Press.

Levy, Luis, et Jeannie Novak. 2010. *Game Development Essentials: Game QA & Testing*. New York : Delmar, Cengage Learning. 260.

Lewis, Ted G. 2006. « High-Level Languages ». In *Computers, Software Engineering, and Digital Devices,* édité par Richard C. Dorf. Florida : CRC Press.

Liang, Y. Daniel, Mohit P. Tahiliani, et Nitk Surathkal. 2014. *Introduction to Programming with C++*. 3e éd. Harlow : Pearson Education. 709.

Liège Game Lab. 2019. « Culture vidéoludique ! » In *Culture vidéoludique !*, 1-15. Liège : Presses Universitaires de Liège.

Limone, Pierpaolo, Benedetta Ragni, et Giusi Antonia Toto. 2023. « The epidemiology and effects of video game addiction: A systematic review and meta-analysis ». *Acta Psychologica* 241 (novembre) : 104047. https://doi.org/10.1016/j.actpsy.2023.104047. [Consulté le 01/03/2024]

Lohmeyer, Eddie. 2023. *Unstable Aesthetics: Game Engines and the Strangeness of Modding*. London : Bloomsbury Academic. 216.

Loiseau, Hugo. 2020. « Chapter 1. The “Science” of Cybersecurity in the Human and Social Sciences: Issues and Reflections ». In *Cybersecurity in Humanities and Social Sciences: A Research Methods Approach,* édité par Hugo Loiseau, Daniel Ventre, et Hartmut Aden, 1-24. Indianapolis : John Wiley & Sons.

Lospinoso, Josh. 2019. *C++ Crash Course: A fast-paced introduction*. San Francisco : No Starch Press. 733.

Lowood, Henry. 2014. « Game Engines and Game History ». In *History of Games International Conference Proceedings 2014*, 179-98.

Maestri, George, Sanford Kennedy, Ralph Frantz, Steve Burke, Jason Greene, Eric Greenleif, Jeremy Hubbell, et al. 1997. *Inside 3D Studio MAX Volume III: Animation*. Indianapolis: New Riders Publishing. 795.

Mahmoud, Dina. 2023. « Hardware Acceleration ». In *Trends in Data Protection and Encryption Technologies*, édité par Valentin Mulder, Alain Mermoud, Vincent Lenders, et Bernhard Tellenbach, 109-14. Cham : Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-33386-6_20. [Consulté le 01/03/2024]

Martin, Paul. 2018. « The Intellectual Structure of Game Research ». *Game Studies* 18 (1). http://gamestudies.org/1801/articles/paul_martin. [Consulté le 01/03/2024]

Marty, Robert. 1990. *L’Algèbre des signes : Essai de sémiotique scientifique d’après C. S. Peirce*. Amsterdam : John Benjamins. 409.

Mathur, Maya B., et Tyler J. VanderWeele. 2019. « Finding Common Ground in Meta-Analysis “Wars” on Violent Video Games ». *Perspectives on Psychological Science* 14 (4) : 705-8. https://doi.org/10.1177/1745691619850104. [Consulté le 01/03/2024]

Mäyrä, Frans. 2009. « Getting into the Game: Doing Multi-Disciplinary Game Studies ». In *The Video Game Theory Reader 2, édité par Mark J.P. Wolf et Bernard Perron*, 313-29. New York : Routledge.

McGrath, Mike. 2021. *Microsoft Assembly x64 Programming: modern coding for MASM, SSE & AVX*. Royal Leamington Spa : In Easy Steps.

McGuire, Morgan, et Odest Chadwicke Jenkins. 2008. *Creating Games: Mechanics, Content, And Technology*. 525.

McShaffry, Mike “MrMike”, et David “Rez” Graham. (2003) 2013. *Game Coding Complete*. 4e éd. Boston. 911.

Melcer, Edward, Truong-Huy Dinh Nguyen, Zhengxing Chen, Alessandro Canossa, Magy Seif El-Nasr, et Katherine Isbister. 2015. « Games Research Today: Analyzing the Academic Landscape 2000-2014 ». In *Proceedings of the 10th International Conference on the Foundations of Digital Games (FDG 2015)*. Pacific Grove, California.

Meunier, Sarah. 2017. « Les recherches sur le jeu vidéo en France. Émergence et enjeux ». *Revue d’anthropologie des connaissances* 11 (3) : 379-96. https://doi.org/10.3917/rac.036.0379. [Consulté le 01/03/2024]

Meunier, Jean-Guy. 2022. *Computational Semiotics*. New York : Bloomsbury Publishing. 276.

Middleton, Martin. 2018. « Thinking About the Data ». In *Behind the Black Box: Sessions with Game Engine Professionals,* édité par Caleb Biasco, Jared Ettinger, Jacob Wilson, Chaojie Zhu, et Yidi Zhu, 78-87. Pittsburgh : ETC Press.

Miller, Robert L., et John D. Brewer. 2003. *The A-Z of Social Research: A Dictionary of Key Social Science Research Concepts*. London : SAGE Publications. 345.

Millington, Ian. (2006) 2019. *Artificial Intelligence for Games*. 3e éd. Florida : CRC Press. 1010.

Mitchell, John C. 2002. *Concepts in Programming Languages*. New York : Cambridge University Press. 530.

Mitchell, Briar Lee. 2012. *Game Design Essentials*. Indianapolis : John Wiley & Sons. 320.

Morisse, Jean-François. 2013. *Histoires secrètes des jeux vidéo : De Tetris à Grand Theft Auto, découvrez les secrets des plus grands hits !* Paris : First Interactive. 280.

Narula, Jitender. 2021. *Implementing Reverse Engineering*. Delhi: BPB Publications. 655.

Neves, Pedro Pinto, et Nelson Zagalo. 2021. « Game Design as an Autonomous Research Subject ». *Information* 12 (9) : 1-22. https://doi.org/10.3390/info12090367. [Consulté le 01/03/2024]

Nicoll, Benjamin, et Brendan Keogh. 2019. *The Unity Game Engine and the Circuits of Cultural Software*. Switzerland : Palgrave macmillan. 123.

Nieborg, David B., et Joke Hermes. 2008. « What is game studies anyway? » *European Journal of Cultural Studies* 11 (2) : 131-47. https://doi.org/10.1177/1367549407088328. [Consulté le 01/03/2024]

Nöth, Winfried. 1990. *Handbook of Semiotics*. Indianapolis : Indiana University Press. 576.

Novak, Jeannie. (2007) 2012. *Game Development Essentials: An Introduction*. 3e éd. New York : Delmar, Cengage Learning. 510.

Nystrom, Robert. 2021. *Crafting Interpreters*. New Jersey : Genever Benning. 626.

Olsen, Jennifer, Andrew Kirmse, et Daniel Sánchez-Crespo. 2001. « Test Drive : On the Open Road with Two of Today's Most Powerful Game Engines ». *Game Developer Magazine* December 2001 : 32-37.

Orianne, Paul. 1983. « L'interdisciplinarité : théorie d'une pratique ou pratique d'une théorie ? » *Revue interdisciplinaire d'études juridiques* 10 (1) : 37-64. https://doi.org/10.3917/riej.010.0037. [Consulté le 01/03/2024]

Pagès, Stéphane. 2019. « Le signe : produit de quoi ? De la diabolique complexité du signe linguistique ». *Signifiances (Signifying)* 2 : 95-110. https://doi.org/10.18145/signifiances.v2i1.175. [Consulté le 01/03/2024]

Paradis, Charles. 2014. « Insert coin to play: Space Invaders and the 100-yen myth ». *Japanese Numismatics* March 2014.

Parberry, Ian. 2001. *Introduction to Computer Game Programming with DirectX 8.0*. Texas : Wordware Publishing. 285.

Paveau, Marie-Anne, et Georges-Elia Sarfati. 2003. *Les grandes théories de la linguistique. De la grammaire comparée à la pragmatique*. Paris : Armand Colin.

Peddie, Jon. 2013. *The History of Visual Magic in Computers: How Beautiful Images are Made in CAD, 3D, VR and AR*. London : Springer-Verlag. 2013.

Peddie, Jon. 2022a. *The History of the GPU 1 Steps to Invention*. Vol. 1. California : Springer Nature. 399.

Peddie, Jon. 2022b. *The History of the GPU 2 Eras and Environment*. Vol. 2. California : Springer Nature. 332.

Peddie, Jon. 2022c. *The History of the GPU 3 New Developments*. Vol. 3. California : Springer Nature. 410.

Pedersen, Roger E. 2003. *Game Design Foundations*. Texas: Wordware Publishing. 378.

Pérez-Castillo, Ricardo, Ignacio García-Rodríguez de Guzmán, et Mario Piattini. 2011. « Architecture-Driven Modernization ». In *Modern Software Engineering Concepts and Practices: Advanced Approaches*, édité par Ali H. Dogru et Veli Biçer, 75-103. Hershey, PA, USA : IGI Global. https://doi.org/10.4018/978-1-60960-215-4.ch004. [Consulté le 01/03/2024]

Perron, Bernard. 2012. *Silent Hill: The Terror Engine*. Michigan: University of Michigan Press. 162.

Perugini, Saverio. 2021. *Programming Languages: Concepts and Implementation*. Ontario: Jones & Bartlett Learning. 840.

Peternier, Achille, Frederic Vexo, et Daniel Thalmann. 2008. « The Mental Vision Framework - A Platform for Teaching, Practicing and Researching with Computer Graphics and Virtual Reality ». In *Transactions on Edutainment I, édité par Zhigeng Pan, Adrian David Cheok, Wolfgang Müller, et Abdennour El Rhabili*, Berlin : Springer-Verlag. 242-260.

Petzold, Charles. 2023. *Code: The Hidden Language of Computer Hardware and Software: Second Edition*. New York : Pearson Education. 469.

Politowski, Cristiano, Fabio Petrillo, João Eduardo Montandon, Marco Tulio Valente, et Yann-Gaël Guéhéneuc. 2021. « Are game engines software frameworks? A three-perspective study ». *Journal of Systems and Software* 171 (Janvier). https://doi.org/10.1016/j.jss.2020.110846. [Consulté le 01/03/2024]

Pranckevičius, Aras. 2018. « Growing Pains in Engine Development ». In *Behind the Black Box: Sessions with Game Engine Professionals*, édité par Caleb Biasco, Jared Ettinger, Jacob Wilson, Chaojie Zhu, et Yidi Zhu, 44-53. Pittsburgh : ETC Press.

Rastier, François. 2001. « Sémiotique et sciences de la culture ». *Linx* 44 : 149-68. https://doi.org/10.4000/linx.1058. [Consulté le 01/03/2024]

Rastier, François. 2010. « Sémiotique et linguistique de corpus ». *Signata* 2010 (1) : 13-38. https://doi.org/10.4000/signata.278. [Consulté le 01/03/2024]

Rastier, François. 2012. « Lire les textes de Saussure ». *Langages* 185 (1) : 7-20. https://doi.org/10.3917/lang.185.0007. [Consulté le 01/03/2024]

Rastier, François. 2017. « De la sémantique structurale à la sémiotique des cultures ». *Actes sémiotiques* 120 : 1-23. https://doi.org/10.25965/as.5734. [Consulté le 01/03/2024]

Rastier, François. (1981) 2022. *Le développement du concept d'isotopie*. Paris : FeniXX réédition numérique (groupe de recherches sémio-linguistiques).

Ravichandran, Kaviyaraj, Mohan Uma, et Soumya Ranjan Jena. 2022. *Introduction To Augmented Reality Hardware: Augmented Reality Will Change The Way We Live Now*. Soumya Ranjan Jena. 219.

Ray, Michael. 2011. *Gaming: From Atari to Xbox (Computing and Connecting in the 21st Century)*. Chicago: Britannica Educational Publishing.

Refenes, Tommy. 2018. « The Engine Sandwich: Made with Super Meat ». In *Behind the Black Box: Sessions with Game Engine Professionals*, édité par Caleb Biasco, Jared Ettinger, Jacob Wilson, Chaojie Zhu, et Yidi Zhu, 88-97. Pittsburgh : ETC Press.

Romero, John. 2023. *Doom Guy: Life in First Person*. New York: Abrams Press. 288.

Romero, Marcos, et Brenden Sewell. (2015) 2022. *Blueprints Visual Scripting for Unreal Engine 5: Unleash the true power of Blueprints to create impressive games and applications in UE5*. 3e éd. Birmingham: Packt Publishing. 547.

Rouse III, Richard. 2005. *Game Design: Theory & Practice, Second Edition*. 2e éd. Texas : Wordware Publishing. 698.

Ryan, Jeff. 2011. *Super Mario: How Nintendo Conquered America*. London : Penguin Books.

Saint, Elizabeth C. 2013. « Les attitudes à l'égard de l'emprunt à l'anglais au Québec et en France : Le cas du domaine informatique ». *Communication, lettres et sciences du langage* 7 (1) : 87-101.

Salen, Katie, et Eric Zimmerman. 2004. *Rules of Play*. Cambridge : MIT Press. 688 pages.

Sanchez, Julio, et Maria P. Canton. 2006. *The PC graphics handbook*. Florida : CRC Press. 1143.

Savchenko, Sergei. 2000. *3D Graphics Programming: Games and Beyond*. Indianapolis : Sams Publishing. 353.

Saussure, Ferdinand. (1916) 2005. *Cours de linguistique générale*, publié par Charles Bally et Albert Sechehaye. Genève : Arbre d'Or. 253.

Scacchi, Walt. 2011. « Modding as an Open Source Approach to Extending Computer Game Systems ». In *Open Source Systems: Grounding Research*, édité par Scott A. Hissam, Barbara Russo, Manoel G. de Mendonça Neto, et Fabio Kon, 62-74. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-24418-6_5. [Consulté le 01/03/2024]

Scott, Michael L. (1999) 2016. *Programming Language Pragmatics*. 4e éd. Boston : Morgan Kaufmann. 956.

Schules, Douglas, Jon Peterson, et Martin Picard. 2018. « Single-Player Computer Role-Playing Games ». In *Role-Playing Game Studies, édité par Sebastian Deterding et José Zagal*. New York : Routledge. 107-129.

Scullion, Chris. 2019. *The NES Encyclopedia: Every game released for the Nintendo Entertainment System*. Philadelphia: White Owl. 270

Scullion, Chris. 2020. *The SNES Encyclopedia: Every game released for the SUPER Nintendo Entertainment System*. Philadelphia: White Owl. 272.

Seacord, Robert C. 2020. *Effective C: An Introduction to Professional C Programming*. San Francisco : No Starch Press. 240.

Sebesta, Robert W. (1989) 2016. *Concepts of Programming Languages*. 11e éd. Harlow: Pearson Education. 758.

Serdar, Adam. 2018. « The Definition and Beginning of a Game Engine ». In *Behind the Black Box: Sessions with Game Engine Professionals, édité par Caleb Biasco, Jared Ettinger, Jacob Wilson, Chaojie Zhu, et Yidi Zhu*, 35-43. Pittsburgh: ETC Press.

Sherif, William. 2015. *Learning C++ by Creating Games with UE4: Learn C++ programming with a fun, real-world application that allows you to create your own games!* Birmingham: Packt Publishing. 322

Sherrod, Allen. 2008. *Game Graphics Programming*. Boston : Course Technology. 645.

Sherrod, Allen, et Wendy Jones. 2012. *Beginning DirectX11 Game Programming*. Boston : Course Technology. 372.

Silberschatz, Abraham, Peter Baer Galvin, et Greg Gagne. (1999) 2008. *Operating System Concepts*. 8e éd. John Wiley & Sons. 971.

Sleep, Drew. 2022. *The History of Video Games*. London : Future Publishing Limited. 143.

Steinhart, Jonathan E. 2019. *The Secret Life of Programs: Understand Computers -- Craft Better Code*. San Francisco : No Starch Press. 465.

Stroustrup, Bjarne. (2013) 2022. *A Tour of C++*. 3e éd. Massachusetts : Pearson Education. 320.

Tan, Wee Hoe. 2019. « A Coaching Framework for Meta-Games: A Case Study of FPS Trainer ». In *Design, Motivation, and Frameworks in Game-Based Learning, édité par Wee Hoe Tan*, 184-213.

Tatsukawa, Kenji. 1995. « Louis Hjelmslev le véritable continuateur de Saussure ». *Linx* 7 : 479-487. https://doi.org/10.4000/linx.1241. [Consulté le 01/03/2024]

Thérien, Gilles. 1989. « Sémiotique et intelligence artificielle ». *Études littéraires* 21 (3) : 67-80. https://doi.org/10.7202/500871ar. [Consulté le 01/03/2024]

Therrien, Carl. 2012. « Video Games Caught Up in History ». In *Before the Crash: Early video game history,* édité par Mark J. P. Wolf, 9-29. Michigan : Wayne State University Press.

Tiercelin, Claudine. 2013. *La pensée-signe : Études sur C. S. Peirce*. Paris : Collège de France. https://books.openedition.org/cdf/2209. [Consulté le 01/03/2024]

Toftedahl, Marcus, et Henrik Engström. 2019. « A Taxonomy of Game Engines and the Tools that Drive the Industry ». In *DiGRA '19, Proceedings of the 2019 DiGRA International Conference: Game, Play and the Emerging Ludo-Mix*.

Toutain, Anne-Gaëlle. 2013. « Entre interprétation et réélaboration : Hjelmslev lecteur du Cours de linguistique générale ». *Dossiers d'HEL* 3 : 1-13.

Vaughan, William. 2012. *[digital] Modeling*. New York : Pearson Education. 410.

Venter, Henk, et Wilhelm Ogterop. 2022. *Unreal Engine 5 Character Creation, Animation, and Cinematics: Create custom 3D assets and bring them to life in Unreal Engine 5 using MetaHuman, Lumen, and Nanite*. Birmingham : Packt Publishing. 584.

Wang, Alf Inge, et Njål Nordmark. 2015. « Software Architectures and the Creative Processes in Game Development ». In *Entertainment Computing - ICEC 2015*, édité par Konstantinos Chorianopoulos, Monica Divitini, Jannicke Baalsrud Hauge, Letizia Jaccheri, et Rainer Malaka, 272-285. Cham : Springer International Publishing. https://doi.org/10.1007/978-3-319-24589-8_21. [Consulté le 01/03/2024]

Wardyga, Brian J. 2019. *The Video Games Textbook: History • Business • Technology*. Florida : CRC Press. 425.

Weiss, Brett. 2007. *Classic Home Video Games, 1972-1984: A Complete Reference Guide*. North Carolina : McFarland.

Whitson, Jennifer R. 2020. « What Can We Learn From Studio Studies Ethnographies?: A “Messy” Account of Game Development Materiality, Learning, and Expertise ». *Games and Culture* 15 (3) : 266-88. https://doi.org/10.1177/1555412018783320. [Consulté le 01/03/2024]

Wilde, Robin. 2023. *Homebrew Game Development and The Extra Lives of Consoles*. Philadelphia : White Owl. 168.

Williams, Andrew. 2017. *History of Digital Games: Developments in Art, Design and Interaction*. New York : Routledge. 271.

Wolf, Mark J. P. 2008a. « Arcade Games of the 1970s ». In *The Video Game Explosion: A History from PONG to PlayStation and Beyond*, édité par Mark J. P. Wolf, 35-44. Connecticut : Greenwood Press.

Wolf, Mark J. P. 2008b. « Arcade Games of the 1990s and Beyond ». In *The Video Game Explosion: A History from PONG to PlayStation and Beyond*, édité par Mark J. P. Wolf, 135-142. Connecticut : Greenwood Press.

Wolf, Mark J. P. 2012a. *Encyclopedia of Video Games: The Culture, Technology, and Art of Gaming*. California : Greenwood Press.

Wolf, Mark J. P. 2012b. « Introduction ». In *Before the Crash: Early video game history,* édité par Mark J. P. Wolf, 1-8. Michigan : Wayne State University Press.

Wolf, Mark J. P., et Bernard Perron. 2003. « Introduction ». In *The Video Game Theory Reader*, édité par Mark J. P. Wolf et Bernard Perron, 1-24. New York : Routledge.

Young, Vaughan. 2005. *Programming a Multiplayer FPS in DirectX*. Hingham : Charles River Media. 486.

Zabban, Vinciane. 2012. « Retour sur les game studies. Comprendre et dépasser les approches formelles et culturelles du jeu vidéo ». *Réseaux* 173-174 (3-4) : 137-176. https://doi.org/10.3917/res.173.0137. [Consulté le 01/03/2024]

Zerbst, Stefan, et Oliver Düvel. 2004. *3D Game Engine Programming*. Boston : Thomson Course Technology. 860.

Zufferey, Sandrine, et Jacques Moeschler. (2010) 2021. *Initiation à la linguistique française*. 3e éd. Malakoff : Armand Colin.

LUDOGRAPHIE

Animal Crossing: *Population: Growing!*. (2001) 2006. Nintendo. Nintendo. GameCube, iQue Player.

Animal Crossing: New Horizons. 2020. Nintendo. Nintendo. Switch.

Ape Escape. 1999. Sony Interactive Entertainment. Sony Interactive Entertainment. PlayStation.

Asteroids. 1979. Atari. Atari. Arcade, Atari 2600.

Batman: Arkham Knight. 2015. Rocksteady Studios. Warner Bros. Interactive Entertainment. PlayStation 4, Windows, Xbox One.

Body Harvest. 1998. Rockstar North. Midway Home Entertainment. Nintendo 64.

Computer Space. 1971. Syzygy Engineering. Nutting Associates. Arcade.

Crash Bandicoot. 1996. Naughty Dog. Sony Interactive Entertainment. PlayStation.

Crash Team Racing. 1999. Naughty Dog. Sony Interactive Entertainment. PlayStation.

Donkey Kong. (1981) 2004. Nintendo. Nintendo. Arcade *et al*.

Donkey Kong Country. (1994) 2003. Rare. Nintendo. Game Boy Advance, Game Boy Color, Super Nintendo.

Donkey Kong Country Returns. 2010. Retro Studios. Nintendo. Nintendo. Wii.

Doom. (1993) 2019. id Software. id Software. MS-DOS et al.

Driver: You Are the Wheelman. (1999) 2009. Ubisoft Reflections. Atari. PlayStation et *al*.

Driver 2: The Wheelman is Back. (2000) 2002. Ubisoft Reflections. Atari. Game Boy Advance, PlayStation, PC.

E.T. the Extra-Terrestrial. 1982. Atari. Atari. Atari 2600.

Fear Effect. 2000. Kronos Digital Entertainment. Square Enix. PlayStation.

Final Fantasy X. 2001. Square Enix. Square Enix. PlayStation 2.

Final Fantasy XIII. 2009. Square Enix. Square Enix. Multi.

Final Fantasy XV. 2016. Square Enix. Square Enix. Multi.

Final Fantasy XVI. 2023. Square Enix. Square Enix. PlayStation 5.

Fortnite. (2017) 2024. Epic Games. Epic Games. Windows *et al*.

Galaxian. 1979. Namco. Midway Manufacturing. Arcade *et al*.

Gears of War. (2006) 2007. Epic Games. Microsoft Game Studio. Windows, Xbox 360.

Gran Turismo. (1997) 1998. Sony Interactive Entertainment. Polys Entertainment Cyberhead. PlayStation.

Grand Theft Auto III. (2001) 2014. Rockstar North. Rockstar Games. PlayStation 2 *et al*.

Grand Theft Auto: San Andreas. (2004) 2015. Rockstar North. Rockstar Games. PlayStation 2 *et al*.

Grand Theft Auto IV. 2008. Rockstar North. Rockstar Games. PlayStation 3, Windows, Xbox 360.

Grand Theft Auto V. (2013) 2022. Rockstar North. Rockstar Games. PlayStation 3 *et al*.

Halo: Combat Evolved Anniversary. (2011) 2020. 343 Industries. Xbox Game Studios. Windows, Xbox 360, Xbox One, Xbox Series X|S.

Halo 2. (2004) 2007. Bungie. Xbox Game Studios. Windows, Xbox.

Harry Potter and the Sorcerer's Stone (2001) 2002. KnowWonder. Windows.

Harry Potter and the Chamber of Secrets. 2002. KnowWonder. Windows.

King's Quest I. (1983) 1990. Sierra On-Line. IBM. IBM PC *et al*.

Legacy of Kain: Soul Reaver. (1999) 2000. Crystal Dynamics. Square Enix. Dreamcast, PlayStation, Windows.

Lightning Returns: Final Fantasy XIII. 2013. Square Enix. Square Enix. PlayStation 3 *et al*.

Mario Kart Tour. (2019) 2024. Nintendo. Nintendo. Android, iOS.

Metal Gear Solid. (1998) 2000. Konami. Konami. PlayStation, Windows.

Metal Gear Solid 2: Sons of Liberty. (2001) 2012. Konami. Konami. PlayStation 2 *et al*.

Metal Gear Solid 2: Substance. (2002) 2003. Konami. Konami. PlayStation 2, Windows, Xbox.

Metal Gear Solid 3: Snake Eater. (2004) 2012. Konami. Konami. PlayStation 2, PlayStation 3, PlayStation Vita, Xbox 360.

Minecraft. (2011) 2018. Mojang. Mojang. Windows *et al*.

Mortal Kombat. Midway Games. Midway Games. Arcade *et al*.

Night Trap. (1992) 2018. Digital Pictures. Sega. Sega CD *et al*.

No Man's Sky. 2016. Hello Games. Hello Games. PlayStation 4 *et al*.

Pac-Man. 1982. Atari. Atari. Atari 2600 *et al*.

Persona 5. (2016) 2017. Atlus. Atlus. PlayStation 3 *et al*.

Pitfall!. 1982. Activision. Activision. Atari 2600 *et al*.

Pong. 1972. Atari. Atari. Arcade *et al*.

Red Dead Redemption 2. (2018) 2019. Rockstar Games. Rockstar Games. PlayStation 4, Windows, Xbox One.

Resident Evil. 1996 (2006). Capcom. Capcom. DS, PlayStation, Saturn, Windows.

Shenmue. (1999) 2018. Sega AM2. Sega. Dreamcast, PlayStation 4, Windows, Xbox One.

Silent Hill. 1999. Konami. Konami. PlayStation.

SpaceWar!. 1962. Steve Russell. Steve Russell. PDP-1.

Splinter Cell. (2002) 2011. Ubisoft. Ubisoft. Xbox *et al*.

Space Invaders. 1978. Taito. Midway. Arcade *et al*.

Street Fighter II: The World Warrior. 1991. Capcom. Capcom. Arcade *et al*.

Super Mario Bros. (1985) 2005. Nintendo. Nintendo. Arcade *et al*.

Super Mario All-Stars. (1993) 2010. Nintendo. Nintendo. Super Nintendo, Wii.

Super Mario 64. (1996) 2003. Nintendo. Nintendo. iQue Player, Nintendo 64.

Super Mario Odyssey. 2017. Nintendo. Nintendo. Switch.

Syphon Filter. 1999. Bend Studio. Sony Computer Entertainment. PlayStation.

Syphon Filter 2. 2000. Bend Studio. Sony Computer Entertainment. PlayStation.

Syphon Filter 3. 2001. Bend Studio. Sony Computer Entertainment. PlayStation.

Tekken. (1994) 2005. Namco. Namco. Arcade, PlayStation.

Tekken 3. (1997) 2005. Namco. Namco. Arcade, PlayStation, PlayStation 2.

Tekken 6. (2007) 2010. Bandai Namco. Bandai Namco. Arcade, PlayStation 3, PlayStation Portable, Xbox 360.

Tekken 7. (2015) 2017. Bandai Namco Entertainment. Bandai Namco Studios. Unreal Engine 4. Arcade, PlayStation 4, Windows, Xbox One.

Tennis for Two. 1958. William Higinbotham. William Higinbotham. Analog Computer.

Tetris. 1984. Alexey Pajitnov. Alexey Pajitnov. Electronika 60, IBM PC.

The Document of Metal Gear Solid 2. (2002) 2003. Konami. Konami. PlayStation 2.

The Legend of Zelda. (1986) 2004. Nintendo. Nintendo. Family Computer Disk System, Game Boy Advance, NES.

The Legend of Zelda: Ocarina of Time. (1998) 2003. Nintendo. Nintendo EAD. GameCube, iQue Player, Nintendo 64.

The Making of Metal Gear Solid 2: Sons of Liberty. 2003. Konami. Konami. PlayStation 2.

Tomb Raider. (1996) 2015. Core Design. Square Enix. PlayStation *et al.*

Tomb Raider: Legend. (2006) 2011. Crystal Dynamics. Square Enix. PlayStation 2 *et al*.

Shadow of the Tomb Raider. (2018) 2019. Crystal Dynamics. Square Enix. PlayStation 4 *et al*.

Uncharted 4: A Thief's End. 2016. Sony Interactive Entertainment. Naughty Dog. PlayStation 4, PlayStation 5, Windows.

Virtua Fighter. (1993) 1996. Sega AM2. Sega. Arcade *et al*.

Watch Dogs 2. 2016. Ubisoft. Ubisoft. PlayStation 4, Windows, Xbox One.

World of Warcraft. (2004) 2005. Blizzard Entertainment. Blizzard Entertainment. macOS, Windows.

MÉDIAGRAPHIE

Adams, Ernest. 2009. « The Designer's Notebook: Sorting Out the Genre Muddle ». *Game Developer*. https://www.gamedeveloper.com/design/the-designer-s-notebook-sorting-out-the-genre-muddle. [Consulté le 01/03/2024]

Annand, Gavin. 2023. « The History of the id Tech Engine ». Superjump. 2023. https://www.superjumpmagazine.com/the-history-of-the-id-tech-engine/. [Consulté le 01/03/2024]

Ashtari, Hossein. 2023. « Machine Language vs. Assembly Language : Key Differences ». Spiceworks. https://www.spiceworks.com/tech/tech-general/articles/machine-vs-assembly-language/. [Consulté le 01/03/2024]

Astle, Aaron. 2023. « 71% of the biggest mobile games use a Unity SDK ». PocketGamer.biz. 2023. https://www.pocketgamer.biz/news/82514/71-of-the-biggest-mobile-games-use-a-unity-sdk/. [Consulté le 01/03/2024]

Autodesk. s. d. « Real Time-Rendering with Autodesk and Unity ». *Autodesk*. https://www.autodesk.com/solutions/real-time-rendering. [Consulté le 01/03/2024]

Bailey, Dustin. 2022. « Steam just reached 50,000 total games listed ». PCGames. https://www.pcgamesn.com/steam/total-games. [Consulté le 01/03/2024]

Batchelor, James. 2021. « How Xbox changed the games industry ». Gamesindustry.biz. 2021. https://www.gamesindustry.biz/how-xbox-changed-the-games-industry. [Consulté le 01/03/2024]

Blender. 2019. « Blender 2.8 Release Note ». Blender. 2019. https://www.blender.org/download/releases/2-80/. [Consulté le 01/03/2024]

Bolding, Jonathan. 2022. « GTA creator reports Rockstar put copyright strikes on his prototype videos ». PC Gamer. https://www.pcgamer.com/gta-creator-reports-rockstar-put-copyright-strikes-on-his-prototype-videos/. [Consulté le 01/03/2024]

Brodkin, Jon. 2013. « How Unity3D Became a Game-Development Beast ». Dice. https://www.dice.com/career-advice/how-unity3d-become-a-game-development-beast. [Consulté le 01/03/2024]

Brown, Stuart. 2019. « The First Video Game ». *Ahoy. YouTube*, 60:00. https://www.youtube.com/watch?v=uHQ4WCU1WQc. [Consulté le 01/03/2024]

Buchanan, Levi. 2016. « Genesis vs. SNES: By the Numbers ». IGN. https://www.ign.com/articles/2009/03/20/genesis-vs-snes-by-the-numbers. [Consulté le 01/03/2024]

Byford, Sam. 2020. « Nintendo Switch overtakes SNES with more than 52 million sold ». The Verge. https://www.theverge.com/2020/1/30/21114839/nintendo-earnings-q3-2019-switch-lifetime-sales. [Consulté le 01/03/2024]

Calvin, Alex. 2020. « How Sony's PlayStation 2 took the world by storm ». Gamesindustry.biz. https://www.gamesindustry.biz/how-sonys-playstation-2-took-the-world-by-storm. [Consulté le 01/03/2024]

Carless, Simon. 2008. « Analysis: What VGChartz Does (And Doesn't) Do For The Game Biz ». Game Developer. https://www.gamedeveloper.com/pc/analysis-what-vgchartz-does-and-doesn-t-do-for-the-game-biz. [Consulté le 01/03/2024]

Caruso, Norman. 2016. « The Story of the Famicom Disk System ». *Gaming Historian. YouTube*, 15:20. https://www.youtube.com/watch?v=r9PuSrn_H1c. [Consulté le 01/03/2024]

Cerny, Mark Evan. 2020. « The Road to PS5 ». *PlayStation. YouTube*, 08:40. https://www.youtube.com/watch?v=ph8LyNIT9sg. [Consulté le 01/03/2024]

Cerny, Mark Evan. 2021. « How PlayStation 5 Was Built (feat. Mark Cerny) ». *Wired. YouTube*, 15:03. https://www.youtube.com/watch?v=cJkx-OLgLzo. [Consulté le 01/03/2024]

Chamberlain, Dan. 2015. « What happened to PlayStation's first indie dev community? » Game Developer. https://www.gamedeveloper.com/design/what-happened-to-playstation-s-first-indie-dev-community-. [Consulté le 01/03/2024]

Chapman, Chris. 2020. « RenderWare: The Engine that Powered an Era ». *Retrohistories. YouTube*, 08:41. https://www.youtube.com/watch?v=FKpDFIWK1DI. [Consulté le 01/03/2024]

CNET Australia. 2008. « Kutaragi: PlayStation 3 "is not a games machine" ». CNET. https://www.cnet.com/tech/gaming/kutaragi-playstation-3-is-not-a-games-machine/. [Consulté le 01/03/2024]

Copetti, Rodrigo. 2023a. « PlayStation 1 Architecture: A Practical Analysis ». Copetti. https://www.copetti.org/projects/consoles/playstation/. [Consulté le 01/03/2024]

Copetti, Rodrigo. 2023b. « Super Nintendo Architecture: A practical analysis ». Copetti. https://www.copetti.org/writings/consoles/super-nintendo/. [Consulté le 01/03/2024]

Cunningham, Zoran. 2012. « How academia is holding back video game criticism ». Game Developer. https://www.gamedeveloper.com/disciplines/how-academia-is-holding-back-video-game-criticism. [Consulté le 01/03/2024]

Dimitris. 2020. « How the Sony PlayStation Net Yaroze DevKit brought Indie Game Development to Consoles | MVG ». *Modern Vintage Gamer. YouTube*, 19 :10. https://www.youtube.com/watch?v=rtE5hmlrcBo. [Consulté le 01/03/2024]

Doucet, Lars. 2020. « The Evolution of Game Genres: NES, Gameboy, and SNES 2 ». Game Developer. https://www.gamedeveloper.com/business/the-evolution-of-game-genres-nes-gameboy-and-snes-2. [Consulté le 01/03/2024]

Dutton, Fred. 2012. « Behind the Classics: Jak & Daxter ». PlayStation Blog. https://blog.playstation.com/archive/2012/08/24/behind-the-classics-jak-daxter/. [Consulté le 01/03/2024]

Dyer, Andy. 2018. « PC Game Mods – From Smurfs to Counter-Strike and Beyond! » *Nvidia*. https://www.nvidia.com/en-us/geforce/news/history-of-pc-game-mods/. [Consulté le 01/03/2024]

Edge. 2008. « The Inside Story of Animal Crossing ». Edge. 2008. https://web.archive.org/web/20131017234121/http://www.edge-online.com/features/inside-story-animal-crossing/. [Consulté le 01/03/2024]

Epic Games. s. d.-a. « Unreal Engine: Previous Versions ». Unreal Engine. https://www.unrealengine.com/en-US/previous-versions. [Consulté le 01/03/2024]

Epic Games. s. d.-b. « Accessing Unreal Engine source code on GitHub ». Unreal Engine. https://www.unrealengine.com/en-US/ue-on-github. [Consulté le 01/03/2024]

Epic Games. 2005. « Unreal Engine 2 ». Unreal Technology. 2005. https://web.archive.org/web/20050826002956/http:/www.unrealtechnology.com/html/technology/ue2.shtml. [Consulté le 01/03/2024]

Epic Games. 2012. « UDK | UnrealScriptReference - UnrealScript Language Reference ». Unreal Engine Docs. 2012. https://docs.unrealengine.com/udk/Three/UnrealScriptReference.html. [Consulté le 01/03/2024]

Epic Games. 2021. « Unreal Engine 4.27 Documentation ». Unreal Engine. https://docs.unrealengine.com/4.27/en-US/. [Consulté le 01/03/2024]

Epic Games. 2023. « Unreal Engine 5.3 Documentation ». Unreal Engine. https://docs.unrealengine.com/5.3/en-US/. [Consulté le 01/03/2024]

Fahs, Travis. 2012. « Exploring the Freescape: How Incentive Software blazed a trail into 3D ». IGN. 2012. https://www.ign.com/articles/2008/10/22/exploring-the-freescape. [Consulté le 01/03/2024]

Forsythe, Alex. 2021. « Blueprints vs. C++: How They Fit Together and Why You Should Use Both ». Awforsythe. https://awforsythe.com/unreal/blueprints_vs_cpp/. [Consulté le 01/03/2024]

GamesIndustry.biz. 2009. « Unreal Development Kit: Free version downloaded over 50,000 times in its first week ». GamesIndustry.biz. 2009. https://www.gamesindustry.biz/unreal-development-kit-free-version-downloaded-over-50-000-times-in-its-first-week. [Consulté le 01/03/2024]

Gibson, Ellie. (2010) 2013. « PS2: The Insiders' Story ». Eurogamer. https://www.eurogamer.net/ps2-the-insiders-story-article. [Consulté le 01/03/2024]

Godot community. 2024. « Godot Engine 4.2 documentation in English ». Godot Docs. 2024. https://docs.godotengine.org/en/stable/index.html.

Goldstein, Maarten. 2001. « Gmax Released ». *Shacknews*. https://www.shacknews.com/article/16177/gmax-released. [Consulté le 01/03/2024]

Goslin, Austen. 2019. « Early access changed everything about games ». Polygon. 2019. https://www.polygon.com/2019/11/13/20961896/decade-of-early-access-minecraft-star-citizen-fortnite. [Consulté le 01/03/2024]

Guillemin, Christophe. 2002. « Un kit Linux pour la Playstation 2, réservé aux initiés ». ZDNET. 2002. https://www.zdnet.fr/actualites/un-kit-linux-pour-la-playstation-2-reserve-aux-inities-2103445.htm. [Consulté le 01/03/2024]

Hall, Charlie. 2017. « Nintendo is using Unreal Engine 4, and Miyamoto says they have it 'mastered' ». Polygon. 2017. https://www.polygon.com/2017/2/7/14533324/nintendo-switch-unreal-engine-miyamoto. [Consulté le 01/03/2024]

Hicks, Jon. 2013. « How the Xbox 360 won the console war ». Techradar. https://www.techradar.com/news/gaming/consoles/how-the-xbox-360-won-the-console-war-1196990. [Consulté le 01/03/2024]

Higgins, Tom. 2009. « Unity 2.5 for Mac and Windows now available! » Unity Blog. https://blog.unity.com/technology/unity-25-for-mac-and-windows-now-available. [Consulté le 01/03/2024]

Higham, Michael. 2023. « Preview: Final Fantasy XVI is a bold new direction for the series ». Ars Technica. https://arstechnica.com/gaming/2023/02/final-fantasy-xvi-hands-on-preview-a-stark-break-from-series-tradition/. [Consulté le 01/03/2024]

Hivner, Brendon. 2003. « Metal Gear Solid 3: Snake Eater (First Look) Preview ». *GamingWorldX*. https://web.archive.org/web/20030704142839/http://gamingworldx.com/ps2/MetalGearSolid3SnakeEaterFirstLook.shtml. [Consulté le 01/03/2024]

Horvath, Stu. 2012. « The Imagination Engine: Why Next-Gen Videogames Will Rock Your World ». Wired. 2012. https://www.wired.com/2012/05/ff-unreal4/. [Consulté le 01/03/2024]

Hugg, Steven. s. d. « Frequently Asked Questions - 8bitworkshop Documentation ». 8bitworkshop. https://8bitworkshop.com/docs/docs/faq.html. [Consulté le 01/03/2024]

Hughes, Colin. 2015. « 20 Years of PlayStation®. Console development past and future - Unite Europe 2015 ». *Unity. YouTube*, 28 :21. https://www.youtube.com/watch?v=fwCVTqSmioI. [Consulté le 01/03/2024]

IndieDB. 2023. « Engines for Games - IndieDB ». *IndieDB*. 2023. https://www.indiedb.com/engines. [Consulté le 01/03/2024]

Ivan, Tom. 2020. « Who is Mark Cerny, the man behind PS5? ». *Video Games Chronicle*. https://www.videogameschronicle.com/features/who-is-mark-cerny/. [Consulté le 01/03/2024]

Jones, Mark. 2006. « Six months to save for PS3 ... if you're game ». Financial Review. https://www.afr.com/technology/six-months-to-save-for-ps3-if-youre-game-20060510-jfgmv. [Consulté le 01/03/2024]

Kent, Steven L. 2000. « PlayStation 2: After the dust settles ». ZDNet.com. https://www.zdnet.com/article/playstation2-after-the-dust-settles-5000106986/. [Consulté le 01/03/2024]

Klepek, Patrick. 2016. « Inside Nintendo's Plan to Save Video Games From Congress ». Vice. https://www.vice.com/en/article/yvw5n7/inside-nintendos-plan-to-save-video-games-from-congress. [Consulté le 01/03/2024]

Konami. 2002. *The Document of Metal Gear Solid 2. PlayStation 2.*[576]

Linneman, John. 2017. « Legacy of Kain: Soul Reaver - the genesis of today's open world tech? » Eurogamer. https://www.eurogamer.net/digitalfoundry-2017-soul-reaver-legacy-of-kain-the-genesis-of-todays-open-world-epics. [Consulté le 01/03/2024]

Linneman, John. 2023. « Tekken 8 tech preview: the first fighting game based on Unreal Engine 5 ». Eurogamer. https://www.eurogamer.net/digitalfoundry-2023-tekken-8-preview-the-first-new-fighting-game-based-on-unreal-engine-5. [Consulté le 01/03/2024]

Loizos, Connie. 2017. « Unity, whose software powers half of all new mobile games, lands $400 million from Silver Lake ». TechCrunch. https://techcrunch.com/2017/05/23/unity-whose-software-powers-half-of-all-new-mobile-games-lands-400-million-from-silver-lake/. [Consulté le 01/03/2024]

Mainelli, Tom. 2000. « The Sony Emotion Engine: Will PlayStation 2 replace your PC? » CNN. https://edition.cnn.com/2000/TECH/computing/02/01/emotion.engine.idg/index.html. [Consulté le 01/03/2024]

Malkia. 2015. « I worked on the port of MGS to PC back in 1999-2000 ». Hacker News. 2015. https://news.ycombinator.com/item?id=9739731. [Consulté le 01/03/2024]

576. Ce document est bien un jeu vidéo. Néanmoins, nous avons décidé de l'inclure au regard des multiples citations présentes dans notre thèse.

McFerran, Damien. 2019. « Talking Point : What Does The Nintendo Seal Of Quality Mean In 2019? » NintendoLife. https://www.nintendolife.com/news/2019/02/talking_point_what_does_the_nintendo_seal_of_quality_mean_in_2019. [Consulté le 01/03/2024]

Morgan, Thomas. 2016. « Face-Off: Batman: Return to Arkham ». Eurogamer. https://www.eurogamer.net/digitalfoundry-2016-batman-return-to-arkham-face-off. [Consulté le 01/03/2024]

NesDev. (2015) 2016. « Were any commercial SNES games written in C? » NesDev. https://forums.nesdev.org/viewtopic.php?t=12330. [Consulté le 01/03/2024]

Neukirchen, Cedric. 2023. « Multiplayer Network Compendium ». Cedric-neukirchen. 2023. https://cedric-neukirchen.net/. [Consulté le 01/03/2024]

Nintendo UK. 2016. « Learn more about the development of the Animal Crossing series in our interview! » Nintendo.co.uk. https://www.nintendo.co.uk/News/2016/November/Learn-more-about-the-development-of-the-Animal-Crossing-series-in-our-interview--1159916.html. [Consulté le 01/03/2024]

O'Grady, Stephen. 2023. « The RedMonk Programming Language Rankings: January 2023 ». RedMonk. 2023. https://redmonk.com/sogrady/2023/05/16/language-rankings-1-23/. [Consulté le 01/03/2024]

Park, Morgan. 2024. « Helldivers 2 was built on an obscure Swedish engine discontinued in 2018: "Our crazy engineers had to do everything with no support" ». PC Gamer. https://www.pcgamer.com/helldivers-2-engine-bitsquid-autodesk-stingray/. [Consulté le 01/03/2024]

Perez, Matt. 2015. « A eulogy for tank controls ». *PC Gamer*. https://www.pcgamer.com/a-eulogy-for-tank-controls/. [Consulté le 01/03/2024]

Plante, Chris. 2012. « Better with age: A history of Epic Games ». Polygon. 2012. https://www.polygon.com/2012/10/1/3438196/better-with-age-a-history-of-epic-games. [Consulté le 01/03/2024]

Racketboy. 2018. « Games that pushed the limits of the Sony Playstation (ps1) ». Racketboy. https://www.racketboy.com/retro/games-that-pushed-the-limits-of-the-sony-playstation-ps1. [Consulté le 01/03/2024]

Racketboy. 2020. « PS1 Strengths and weaknesses vs N64 and Sega Saturn ». Racketboy. http://www.racketboy.com/journal/ps1-strength-and-weaknesses-vs-n64-sega-saturn. [Consulté le 01/03/2024]

Reinhart, Brandon. 2000. « Postmortem: Epic Games' Unreal Tournament ». Game Developer. https://www.gamedeveloper.com/programming/postmortem-epic-games-i-unreal-tournament-i-. [Consulté le 01/03/2024]

Retro Reversing. 2020a. « Official PlayStation 1 Development Kit (Hardware) ». Retro Reversing. https://www.retroreversing.com/official-playStation-devkit. [Consulté le 01/03/2024]

Retro Reversing. 2020b. « SNES (Super Famicom) Development Kit Hardware ». Retro Reversing. https://www.retroreversing.com/super-famicom-snes-development-kit/. [Consulté le 01/03/2024]

Reynolds, Matthew. 2013. « Naughty Dog will use existing Uncharted, The Last of Us engine for PS4 ». DigitalSpy. 2013. https://www.digitalspy.com/videogames/ps4/a485922/naughty-dog-will-use-existing-uncharted-the-last-of-us-engine-for-ps4/. [Consulté le 01/03/2024]

Sanchez, Léa, Raphaëlle Aubert, Pierre Trouvé, et Corentin Lamy. 2021. « PlayStation 5 et Xbox Series X : après un an, quel bilan pour les consoles "nouvelle génération" ». Le Monde. https://www.lemonde.fr/pixels/article/2021/11/19/playstation-5-et-xbox-series-x-apres-un-an-quel-bilan-pour-les-consoles-nouvelle-generation_6102888_4408996.html. [Consulté le 01/03/2024]

Scarpazza, Daniele Paolo, Oreste Villa, et Fabrizio Petrini. 2007. « Programming the Cell Processor ». *Dr. Dobb's*. https://www.drdobbs.com/parallel/programming-the-cell-processor/197801624. [Consulté le 01/03/2024]

Shuman, Sid. 2012. « Behind the Classics – Legacy of Kain: Soul Reaver ». PlayStation Blog. https://blog.playstation.com/archive/2012/10/12/behind-the-classics-legacy-of-kain-soul-reaver/. [Consulté le 01/03/2024]

Sinclair, Brendan. 2014. « Should developers go to school? Participants from across the industry offer eight different answers to one though question ». Gamesindustry.biz. https://www.gamesindustry.biz/articles/2014-04-22-should-developers-go-to-school. [Consulté le 01/03/2024]

Smith, Tony. 2000. « Sony adds Basic to PlayStation to sidestep EC import tax ». *The Register*. https://www.theregister.com/2000/11/07/sony_adds_basic_to_playstation/. [Consulté le 01/03/2024]

Stanton, Rich. 2015. « What made Super Mario 64 so special? » *Eurogamer*. https://www.eurogamer.net/what-made-super-mario-64-so-special. [Consulté le 01/03/2024]

SteamDB. 2024. « What are games built with and what technologies do they use? » SteamDB. https://steamdb.info/tech/. [Consulté le 01/03/2024]

Sulic, Ivan. (2001) 2012. « Harry Potter and the Sorcerer's Stone ». IGN. https://www.ign.com/articles/2001/08/09/harry-potter-and-the-sorcerers-stone-12. [Consulté le 01/03/2024]

Swan, Cameron. 2022. « The Biggest Controversies in Mortal Kombat History. Since its debut in 1992 ». Gamerant. https://gamerant.com/mortal-kombat-controversy-history-congress-violence/. [Consulté le 01/03/2024]

Szczepaniak, John. 2012. « 15 Years Later: How Sony's Net Yaroze Kickstarted Indie Console Development ». Game Developer. https://www.gamedeveloper.com/disciplines/15-years-later-how-sony-s-net-yaroze-kickstarted-indie-console-development. [Consulté le 01/03/2024]

Tach, Dave. 2014. « The new Unreal Tournament will be free, moddable and developed with players ». Polygon. 2014. https://www.polygon.com/2014/5/8/5695404/unreal-tournament-new-2014-free-unreal-engine-4-epic-games. [Consulté le 01/03/2024]

Tagbar, Chad. 2023. « Tekken 8 And All You Need To Know About It ». KJC Esports. https://www.kjcesports.com/feature/tekken-8-and-all-you-need-to-know-about-it/. [Consulté le 01/03/2024]

The Game Creators. s. d. « Dark Basic Pro Open Source ». The Game Creators. https://www.thegamecreators.com/product/dark-basic-pro-open-source. [Consulté le 01/03/2024]

Thomsen, Mike. 2012. « History of the Unreal Engine ». IGN. 2012. https://www.ign.com/articles/2010/02/23/history-of-the-unreal-engine. [Consulté le 01/03/2024]

TIOBE. 2023. « TIOBE Index for December 2023 ». TIOBE. 2023. https://www.tiobe.com/tiobe-index/. [Consulté le 01/03/2024]

TurboSquid. s. d. « I haven't received a Gmax registration code ». Turbosquid. https://support.turbosquid.com/hc/en-us/articles/230095687. [Consulté le 01/03/2024]

Unity Technologies. 2023. « Unity Documentation - Version 2023.3 ». Unity Documentation. https://docs.unity3d.com/2023.2/Documentation/Manual/index.html. [Consulté le 01/03/2024]

Unity Technologies. s. d.-a « Real-time Rendering in 3D ». Unity. https://unity.com/how-to/real-time-rendering-3d. [Consulté le 01/03/2024]

Unity Technologies. s. d.-b « Unity Source Code ». Unity. https://unity.com/products/source-code. [Consulté le 01/03/2024]

Unseen64. 2006. « Final Fantasy X [10] Original Trailer [BETA - Proto 2000] ». 2:30. YouTube. https://www.youtube.com/watch?v=72vckLXI7x4. [Consulté le 01/03/2024]

Valich, Theo. 2008. « Tim Sweeney, Part 3: Unreal Engine 4.0 aims at next-gen console war ». TG Daily. 2008. https://web.archive.org/web/20100414203822/http://www.tgdaily.com/business-and-law-features/36436-tim-sweeney-part-3-unreal-engine-40-aims-at-next-gen-console-war. [Consulté le 01/03/2024]

Walton, Brett. 2010. « Editorial: Why it is so Easy to Blame VGChartz – News ». VGChartz. https://www.vgchartz.com/article/82746/editorial-why-it-is-so-easy-to-blame-vgchartz/. [Consulté le 01/03/2024]

Webb, Kevin. 2020. « The last known Nintendo PlayStation prototype is up for auction — here's the wild story behind the one-of-a-kind, doomed collaboration with Sony ». Business Insider. https://www.businessinsider.com/nintendo-playstation-prototype-auction-history-sale-2020-1?r=US&IR=T. [Consulté le 01/03/2024]

Wright, Gavin. 2022. « agnostic ». TechTarget. https://www.techtarget.com/whatis/definition/agnostic. [Consulté le 01/03/2024]

White, Stephen. 2002. « Postmortem: Naughty Dog's Jak and Daxter: the Precursor Legacy ». Game Developer. https://www.gamedeveloper.com/design/postmortem-naughty-dog-s-i-jak-and-daxter-the-precursor-legacy-i-. [Consulté le 01/03/2024]

Wilkinson, Oli. 2009. « Unity CEO David Helgason ». Gamedev.net. 2009. https://www.gamedev.net/tutorials/industry/interviews/unity-ceo-david-helgason-r2722/. [Consulté le 01/03/2024]

Xbox. 2021a. « Power On: The Story of Xbox | Chapter 1: The Renegades ». *Xbox,* YouTube, 41:02. https://www.youtube.com/watch?v=AJYsA1jXf60. [Consulté le 01/03/2024]

Xbox. 2021b. « Power On: The Story of Xbox | Chapter 2: The Valentine's Day Massacre ». *Xbox,* YouTube, 39:09. https://www.youtube.com/watch?v=yT_i6hXf9WU. [Consulté le 01/03/2024]

LISTE DES ILLUSTRATIONS

FIGURES

Figure 1. Axes fondamentaux de la démarche scientifique (Loiseau 2020, 9)_____ 27

Figure 2. Classes des *Actors* dans Unreal Engine 4 _____ 256

TABLEAUX

Tableau 1. Conventions des citations de la ludographie IX

Tableau 2. Conventions des références de la ludographie X

Tableau 3. Ventes mondiales de la première et deuxième génération de consoles..... 14

Tableau 4. Genres des 30 jeux NES de la série *Black Box* (Doucet 2020) 20

Tableau 5. Ébauche de la problématique de recherche 24

Tableau 6. Problématique de la recherche doctorale 25

Tableau 7. Hypothèse de départ de la recherche doctorale 28

Tableau 8. Corpus développemental direct de la recherche 34

Tableau 9. Méthodologies des *Game Studies* (Egenfeldt-Nielsen *et al.* 2015, 11) 53

Tableau 10. Problématique, hypothèse et corpus de la recherche doctorale 76

Tableau 11. Étapes du module de scientificité 87

Tableau 12. Différents modules technologiques du DLT 90

Tableau 13. Ventes mondiales de la cinquième génération (Wardyga 2019, 212)..... 133

Tableau 14. Genèse des API graphiques principales (Peddie 2022b, 203-215) 147

Tableau 15. Genèse des moteurs de jeu en 2.5D et 3D 152

Tableau 16. Définition de la notion de mécanisme interactif 158

Tableau 17. Définition du signe virtuel interactif haut niveau..... 160

Tableau 18. Exemples de signes virtuels interactifs haut niveau..... 160

Tableau 19. Définition du signe virtuel interactif bas niveau 162

Tableau 20. Langages utilisés entre la troisième et la cinquième génération 166

Tableau 21. Exemples de vecteurs de standardisation 171

Tableau 22. Interactants disponibles dans le corpus vidéoludique 176

Tableau 23. Nomenclature des environnements du corpus vidéoludique 180

Tableau 24. Typologie des niveaux dans différents moteurs de jeu..... 182

Tableau 25. Ventes mondiales de la sixième génération 193

Tableau 26. Trois classements de popularité des langages de programmation..... 197

Tableau 27. Classement des moteurs selon Pedersen (2003)..... 202

Tableau 28. Immutabilité interactive 220
Tableau 29. Synthèse de mutabilité, immutabilité et ancrage générationnel..... 222
Tableau 30. Faisceaux et substrats virtuels..... 231
Tableau 31. Ventes mondiales de la septième génération (Wardyga 2019, 318)..... 241
Tableau 32. Typologie des trois interactants virtuels sur *Unreal Engine 4*..... 254
Tableau 33. Concepts du niveau sémique interactif 265
Tableau 34. Ventes mondiales de la huitième génération..... 275
Tableau 35. Outils de développement listés par SteamDB (2024)..... 279
Tableau 36. Genres vidéoludiques de la boutique en ligne Epic Games Store 362
Tableau 37. Genres vidéoludiques de la boutique en ligne GOG.com 363
Tableau 38. Genres vidéoludiques de la boutique en ligne Microsoft Store 363
Tableau 39. Genres vidéoludiques de la boutique en ligne Nintendo eShop..... 363
Tableau 40. Genres vidéoludiques de la boutique en ligne PlayStation Store 364
Tableau 41. Genres vidéoludiques du rapport *Unity Gaming Report 2023*..... 364
Tableau 42. Genres vidéoludiques de la boutique en ligne Steam..... 365
Tableau 43. Ancien corpus vidéoludique 368

PAGES ANNEXES

ANNEXE A : SCHÉMAS ORIGINAUX

Cette section indique les différents schémas en version originale, par opposition à la version traduite disponible dans notre thèse.

FIGURE 1

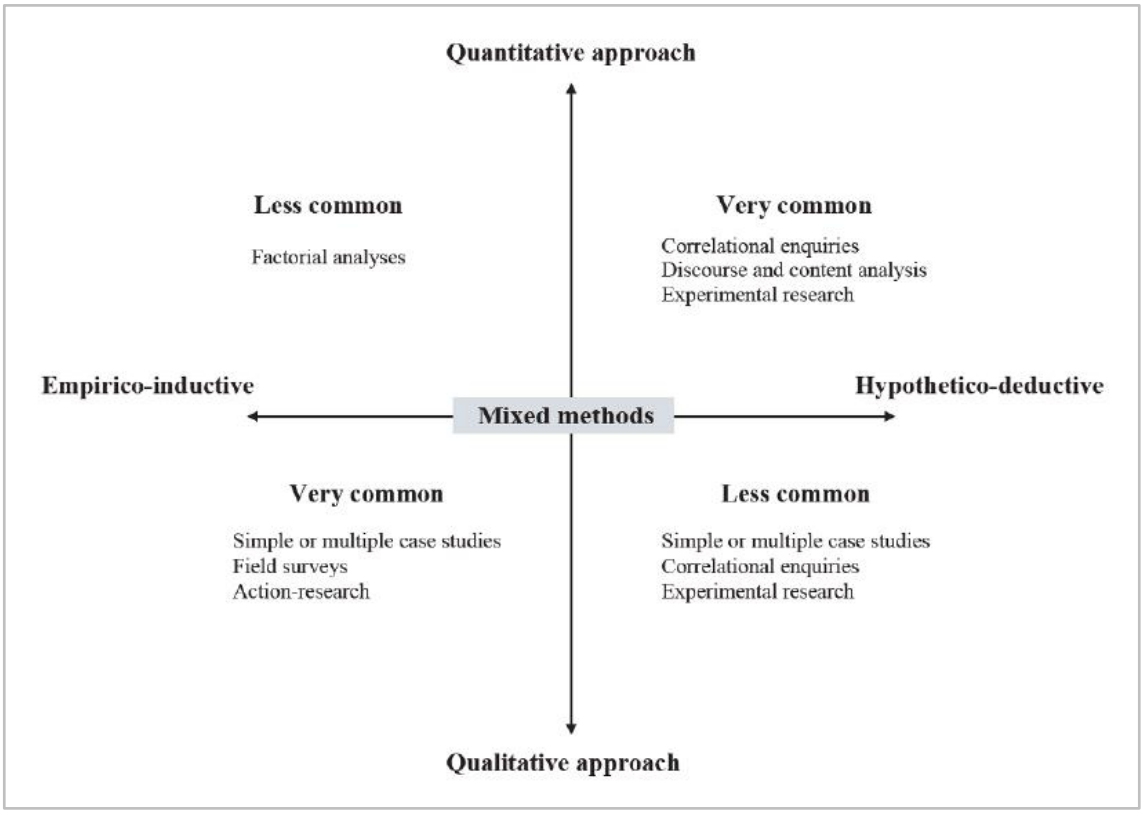


Figure: Quantitative and qualitative approaches and empirico-inductive and hypothetico-deductive approaches in the human and social sciences (Loiseau 2020, 9).

ANNEXE B : NOUVEAU CORPUS VIDÉOLUDIQUE

Ce corpus vidéoludique fut élaboré à la fin de la thèse doctorale, après avoir effectué l'ensemble des recherches. Il s'agit donc d'une liste rédigée a posteriori.

5E GÉNÉRATION DE CONSOLES

007: Tomorrow Never Dies.

Ape Escape.

Body Harvest.

Crash Bandicoot 2: Cortex Strikes Back.

Crash Team Racing.

DOOM.

Driver 2: The Wheelman is Back.

Fear Effect.

Gran Turismo 2.

Harry Potter and the Sorcerer's Stone.

Legacy of Kain: Soul Reaver.

Metal Gear Solid.

Quake.

Quake II.

Silent Hill.

Syphon Filter.

Syphon Filter 2.

Tekken 3.

The Unholy War.

Unreal (1998).

6E GÉNÉRATION DE CONSOLES

Animal Crossing: Population Growing!

Grand Theft Auto III.

Grand Theft Auto: San Andreas.

Metal Gear Solid 2: Sons of Liberty.

Metal Gear Solid 3: Snake Eater.

Tom Clancy's Splinter Cell.

7E GÉNÉRATION DE CONSOLES

Grand Theft Auto IV.

8E GÉNÉRATION DE CONSOLES

Uncharted 4: A Thief's End.

ANNEXE C : PRÉSENCE DES GENRES VIDÉOLUDIQUES

Date de dernière actualisation des données : 14 janvier 2024.

Cette section regroupe les différents genres vidéoludiques utilisés dans plusieurs boutiques de jeux vidéo en ligne (*Epic Games Store*, GOG, *Microsoft Store, Nintendo eShop*, *PlayStation Store* et *Steam*), ainsi que dans des publications pour des développeurs de jeux vidéo (*Unity Gaming Report 2023*). Pour chaque liste, nous choisissons d'indiquer les occurrences uniquement en anglais, car l'ensemble des outils de développement de jeux vidéo 3D sont également analysés en anglais. Par ailleurs, nous utilisons une virgule pour séparer chaque genre.

EPIC GAMES STORE

Les genres vidéoludiques de la boutique *Epic Games Store* sont mentionnés dans la section *Browse* (en français « Parcourir »), et sont utilisés comme des filtres pour classer les différents jeux.

URL	Epic Games. 2023. Epic Games Store. https://store.epicgames.com/en-US/browse?sortBy=releaseDate&sortDir=DESC&category=Game&count=40.
Genres	*Action, Action-Adventure, Adventure, Card Game, Casual, City Builder, Comedy, Dungeon Crawler, Exploration, Fantasy, Fighting, First Person, Horror, Indie, MOBA, Music, Narration, Open World, Party, Platformer, Puzzle, Racing, Retro, Rogue-Lite, RPG, RTS, Shooter, Simulation, Space, Sports, Stealth, Strategy, Survival, Tower Defense, Trivia, Turn-Based, Turn-Based Strategy.*

Tableau 36. Genres vidéoludiques de la boutique en ligne Epic Games Store

GOG

Les genres vidéoludiques de la boutique en ligne *GOG*, exploités comme des filtres, sont mentionnés à deux endroits distincts : la barre de navigation du site ainsi que le menu. On observe toutefois des différences entre la barre de navigation et le menu.

URL	CD Projekt. 2023. GOG.com. https://www.gog.com/en/games?query=the&order=desc:score.
Genres	*Action, Adventure, Racing, Role-playing, Shooter, Simulation, Sports, Strategy.*
Genres (Menu)	*RPG, Action, Adventure, Strategy, Open world, Indie, Shooters, Platformers, City builders.*

Tableau 37. Genres vidéoludiques de la boutique en ligne GOG.com

MICROSOFT STORE

Les genres vidéoludiques de la plate-forme *Microsoft Store*, agissant comme des filtres, sont disponibles dans la section *Browse all games* (en français « Parcourir tous les jeux »).

URL	Microsoft. 2023. Microsoft Store. https://www.xbox.com/en-US/browse/games.
Genres	*Action & adventure, Card & board, Casino, Classics, Companion, Educational, Family & kids, Fighting, Multi-player online battle arena, Music, Other, Platformer, Puzzle & trivia, Racing & flying, Role playing, Shooter, Simulation, Social, Sports, Strategy, Tools, Video, Word.*

Tableau 38. Genres vidéoludiques de la boutique en ligne Microsoft Store

NINTENDO ESHOP

Les genres vidéoludiques de la plateforme Nintendo eShop sont utilisés comme étant des filtres.

URL	Nintendo. 2023. Nintendo eShop. https://www.nintendo.com/search/?slug=nintendo-switch-games#p=1&cat=gme&sort=df.
Genres	*Action, Adventure, Arcade, Board Game, Education, Fighting, First-Person, Lifestyle, Multiplayer, Music, Other, Party, Platformer, Puzzle, Racing, Role-Playing, Simulation, Sports, Strategy, Training.*

Tableau 39. Genres vidéoludiques de la boutique en ligne Nintendo eShop

PLAYSTATION STORE

Les genres vidéoludiques de la boutique en ligne *PlayStation Store* sont disponibles dans la section *Browse* (en français « Parcourir ») et utilisés comme des filtres.

URL	Sony Interactive Entertainment. 2023. PlayStation Store. https://store.playstation.com/en-us/pages/browse.
Genres	*Action, Adventure, Arcade, Role Playing Games, Puzzle, Shooter, Casual, Simulation, Strategy, Unique, Driving/ Racing, Sport, Family, Horror, Fighting, Party, Simulator, Music/ Rhythm, Brain Training, Educational, Quiz, Adult.*

Tableau 40. Genres vidéoludiques de la boutique en ligne PlayStation Store

UNITY REPORT 2023

Dans le rapport annuel de l'entreprise *Unity*, développeurs du moteur éponyme *Unity*, nous pouvons voir les différents genres vidéoludiques qui apparaissent dans trois pages différentes : p. 18, p. 32 et p. 40. La page 18 concerne la popularité des genres en fonction de la taille d'un studio de développement, la page 32 est liée aux revenus générés par les publicités et achats *In-App*, enfin la page 40 est liée à la longévité des jeux vidéo.

URL	Unity. 2023. *Unity Gaming Report 2023*. https://create.unity.com/gaming-report.
***Genres* p. 18**	*Action, Adventure, RPG, Puzzle, Simulation, Shooter, Arcade, Strategy, Platformer, Board/ Card, Racing + Sport, Massively Multiplayer + Battle Arena, Fighting, Other.*
Genres p. 32 et p. 40	*Action, Adventure, Arcade, Board, Card, Casino, Casual, Hypercasual, Puzzle, Racing, Role Playing, Simulation, Sports, Strategy, Trivia, Word.*

Tableau 41. Genres vidéoludiques du rapport *Unity Gaming Report 2023*

VALVE STEAM

Les genres vidéoludiques de la boutique virtuelle Steam sont directement indiqués dans des *Tags*. Parce qu'il en existe une infinité, nous choisissons de répertorier seulement les vingt-cinq *Tags* les plus populaires.

URL	Valve. 2023. Steam. https://store.steampowered.com/tag/browse#global_492.
Genres	*Indie, Action, Adventure, Casual, RPG, Simulation, Strategy, Singleplayer, Early Access, Free to Play, 2D, 3D, Atmospheric, Fantasy, Story Rich, Colorful, Multiplayer, Puzzle, Pixel Graphics, Exploration, Massively Multiplayer, Sports, Cute, Violent, First-Person.*

Tableau 42. Genres vidéoludiques de la boutique en ligne Steam

ANNEXE D : CORPUS DÉVELOPPEMENTAL INDIRECT

Ce corpus vidéoludique fut élaboré à la fin de la thèse doctorale, après avoir effectué l'ensemble des recherches. Il s'agit donc d'une liste rédigée *a posteriori*. Naturellement, celle-ci n'indique pas les éléments du corpus développemental direct, dont la liste est intégrée dans l'introduction.

LANGAGES DE PROGRAMMATION

Assembleurs.

BASIC.

DarkBasic.

C.

Kismet.

UnrealScript.

YA-Basic.

MOTEURS DE JEU, MOTEURS GRAPHIQUES, API ET KIT

Adventure Game Interpreter.

Autodesk Stingray (*Bitsquid*).

Blender Game Engine.

Build Engine.

Catacomb 3-D Engine.

Crystal Space.

DarkBasic Pro.

DirectX.

DOOM Engine.

Freescape Engine.

Gold Box engine.

IRIS-GL 1.0.

Irrlicht.

Keen Engine.

Linux Kit.

Net Yaroze.

OGRE.

OpenGL.

PhyreEngine.

Quake Engine.

Quake II Engine.

RenderWare Platform.

RenderWare Studio.

Tektronix Plot 10.

Unity.

Unreal Development Kit.

Unreal Engine.

Unreal Engine 2.

Unreal Engine 3.

Unreal Engine 5.

UPBGE.

Wolfenstein 3D Engine.

ANNEXE E : ANCIEN CORPUS VIDÉOLUDIQUE

Cet ancien corpus vidéoludique est obsolète et n'est plus utilisé pour cette recherche. Néanmoins, il est conservé pour expliciter certains choix, notamment au regard du corpus vidéoludique non défini à l'avance et indiqué en Annexe B. Dans tous les cas, cet ancien corpus intègre un classement selon les multiples générations de consoles. Notre choix s'est porté sur une diversité dans les différents genres vidéoludiques. En voici la liste : action, aventure, combat, course, party, platform, puzzle, rôle, sport, stratégie, simulation et tir. Nous avions également pris en compte la diversité graphique des jeux et aussi incorporé des œuvres qui sont des adaptations de livres ou de films. Enfin, les normes de présentation de ce corpus sont indiquées dans le tableau ci-dessous.

Type	Norme
Ancien Corpus vidéoludique	Nom du jeu (américain si plusieurs). Année de publication. Console de la version utilisée. Nombre mondial de ventes. Site de référence des ventes.

Tableau 43. Ancien corpus vidéoludique

5E GÉNÉRATION DE CONSOLES

007: Tomorrow Never Dies. 1999. PlayStation. Plus de 3,2 millions d'unités vendues. https://www.vgchartz.com/game/6508/007-tomorrow-never-dies/?region=All. [Consulté le 01/03/2024]

A Bug's Life. 1998. PlayStation. Plus de 2,7 millions d'unités vendues. https://www.vgchartz.com/game/6192/a-bugs-life/?region=All. [Consulté le 01/03/2024]

Ape Escape. 1999. PlayStation. Plus de 1,6 million d'unités vendues. https://www.vgchartz.com/game/86/ape-escape/?region=All. [Consulté le 01/03/2024]

Colin McRae Rally. 2000. PlayStation. Plus de 2,8 millions d'unités vendues. https://www.vgchartz.com/game/6208/colin-mcrae-rally/?region=All. [Consulté le 01/03/2024]

Crash Bandicoot 3: Warped. 1998. PlayStation. Plus de 7,1 millions d'unités vendues. https://www.vgchartz.com/game/408/crash-bandicoot-3-warped/?region=All. [Consulté le 01/03/2024]

Crash Team Racing. 1999. PlayStation. Plus de 4,7 millions d'unités vendues. https://www.vgchartz.com/game/413/crash-team-racing/?region=All. [Consulté le 01/03/2024]

Croc: Legend of the Gobbos. 1997. PlayStation. Plus de 3,5 millions d'unités vendues. https://www.vgchartz.com/game/426/croc-legend-of-the-gobbos/?region=All. [Consulté le 01/03/2024]

Digimon World. 2000. PlayStation. Plus de 1,3 million d'unités vendues. https://www.vgchartz.com/game/522/digimon-world/?region=All. [Consulté le 01/03/2024]

Driver 2: The Wheelman is Back. 2000. PlayStation. Plus de 4,7 millions d'unités vendues. https://www.vgchartz.com/game/591/driver-2/?region=All.

Final Fantasy IX. 2000. PlayStation. Plus de 5,5 millions d'unités vendues. https://www.vgchartz.com/game/752/final-fantasy-ix/?region=All. [Consulté le 01/03/2024]

GoldenEye 007. 1997. Nintendo 64. Plus de 8 millions d'unités vendues. https://www.vgchartz.com/game/877/goldeneye-007/?region=All. [Consulté le 01/03/2024]

Gran Turismo. 1998. PlayStation. Plus de 10,8 millions d'unités vendues. https://www.vgchartz.com/game/884/gran-turismo/?region=All. [Consulté le 01/03/2024]

Harry Potter and the Sorcerer's Stone. 2001. PlayStation. Plus de 3,7 millions d'unités vendues. https://www.vgchartz.com/game/6293/harry-potter-and-the-sorcerers-stone/?region=All. [Consulté le 01/03/2024]

Intelligent Qube. 1997. PlayStation. Plus de 1,2 million d'unités vendues. https://www.vgchartz.com/game/3511/intelligent-qube/?region=All. [Consulté le 01/03/2024]

Mario Party 3. 2001. Nintendo 64. Plus de 1,9 million d'unités vendues. https://www.vgchartz.com/game/1258/mario-party-3/?region=All. [Consulté le 01/03/2024]

Metal Gear Solid. 1998. PlayStation. Plus de 6 millions d'unités vendues. https://www.vgchartz.com/game/1325/metal-gear-solid/?region=All. [Consulté le 01/03/2024]

Parasite Eve. 1998. PlayStation. Plus de 1,9 million d'unités vendues. https://www.vgchartz.com/game/1717/parasite-eve/?region=All. [Consulté le 01/03/2024]

Perfect Dark. 2000. Nintendo 64. Plus de 2,5 millions d'unités vendues. https://www.vgchartz.com/game/1724/perfect-dark/?region=All. [Consulté le 01/03/2024]

Pokémon Stadium 2. 2001. Nintendo 64. Plus de 2,5 millions d'unités vendues. https://www.vgchartz.com/game/1768/pokemon-stadium-2/?region=All. [Consulté le 01/03/2024]

Silent Hill. 1999. PlayStation. Plus de 1,6 million d'unités vendues. https://www.vgchartz.com/game/2051/silent-hill/?region=All. [Consulté le 01/03/2024]

Spider-Man. 2000. PlayStation. Plus de 3,1 millions d'unités vendues. https://www.vgchartz.com/game/6430/spider-man/?region=All. [Consulté le 01/03/2024]

Star Fox 64. 1997. Nintendo 64. Plus de 4 millions d'unités vendues. https://www.vgchartz.com/game/2219/star-fox-64/?region=All. [Consulté le 01/03/2024]

Star Wars: Shadow of the Empire. 1996. Nintendo 64. Plus de 2,6 millions d'unités vendues. https://www.vgchartz.com/game/2215/star-wars-shadows-of-the-empire/?region=All. [Consulté le 01/03/2024]

Super Mario 64. 1996. Nintendo 64. Plus de 11,9 millions d'unités vendues. https://www.vgchartz.com/game/2278/super-mario-64/?region=All. [Consulté le 01/03/2024]

Super Smash Bros. 1996. Nintendo 64. Plus de 5,5 millions d'unités vendues. https://www.vgchartz.com/game/2290/super-smash-bros/?region=All. [Consulté le 01/03/2024]

Syphon Filter 2. 2000. PlayStation. Plus de 2,1 millions d'unités vendues. https://www.vgchartz.com/game/2312/syphon-filter-2/?region=All. [Consulté le 01/03/2024]

Tekken 3. 1998. PlayStation. Plus de 8,3 millions d'unités vendues. https://www.vgchartz.com/game/2344/tekken-3/?region=All. [Consulté le 01/03/2024]

The Legend of Zelda: Ocarina of Time. 1998. Nintendo 64. Plus de 7, 6 millions d'unités vendues. https://www.vgchartz.com/game/2399/the-legend-of-zelda-ocarina-of-time/?region=All. [Consulté le 01/03/2024]

Tomb Raider. 1996. PlayStation. Plus de 4,6 millions d'unités vendues. https://www.vgchartz.com/game/2472/tomb-raider/?region=All. [Consulté le 01/03/2024]

Tony Hawk's Pro Skater. 1999. PlayStation. Plus de 5 millions d'unités vendues. https://www.vgchartz.com/game/5930/tony-hawks-pro-skater/?region=All. [Consulté le 01/03/2024]

6E GÉNÉRATION DE CONSOLES

Animal Crossing: Population: Growing!. 2002. GameCube. Plus de 2,2 millions d'unités vendues. https://www.vgchartz.com/game/81/animal-crossing/?region=All. [Consulté le 01/03/2024]

Dark Cloud. 2001. PlayStation 2. Plus de 1,5 million d'unités vendues. https://www.vgchartz.com/game/3051/dark-cloud/?region=All. [Consulté le 01/03/2024]

Fable. 2004. Xbox. Plus de 3 millions d'unités vendues. https://www.vgchartz.com/game/703/fable/?region=All. [Consulté le 01/03/2024]

Final Fantasy X. 2001. PlayStation 2. Plus de 8,6 millions d'unités vendues. https://www.vgchartz.com/game/759/final-fantasy-x/?region=All. [Consulté le 01/03/2024]

Gran Turismo 4. 2005. PlayStation 2. Plus de 11,7 millions d'unités vendues. https://www.vgchartz.com/game/887/gran-turismo-4/?region=All. [Consulté le 01/03/2024]

Grand Theft Auto: San Andreas. 2004. PlayStation 2. Plus de 17,3 millions d'unités vendues. https://www.vgchartz.com/game/3427/grand-theft-auto-san-andreas/?region=All. [Consulté le 01/03/2024]

Halo 2. 2004. Xbox. Plus de 8 millions d'unités vendues. https://www.vgchartz.com/game/940/halo-2/?region=All. [Consulté le 01/03/2024]

Harry Potter and the Chamber of Secrets. 2002. PlayStation 2. Plus de 2,6 millions d'unités vendues. https://www.vgchartz.com/game/19251/harry-potter-and-the-chamber-of-secrets/?region=All. [Consulté le 01/03/2024]

Kingdom Hearts II. 2006. PlayStation 2. Plus de 5,2 millions d'unités vendues. https://www.vgchartz.com/game/1130/kingdom-hearts-ii/?region=All. [Consulté le 01/03/2024]

Mario Kart: Double Dash. 2003. GameCube. 6,8 millions d'unités vendues. https://www.vgchartz.com/game/1255/mario-kart-double-dash/?region=All. [Consulté le 01/03/2024]

Mario Party 7. 2005. GameCube. Plus de 2 millions d'unités vendues. https://www.vgchartz.com/game/1262/mario-party-7/?region=All. [Consulté le 01/03/2024]

Metal Gear Solid 2: Sons of Liberty. 2001. PlayStation 2. Plus de 6 millions d'unités vendues. https://www.vgchartz.com/game/3774/metal-gear-solid-2-sons-of-liberty/?region=All. [Consulté le 01/03/2024]

Metal Gear Solid 3: Snake Eater. 2004. PlayStation 2. Plus de 4,3 millions d'unités vendues. https://www.vgchartz.com/game/1328/metal-gear-solid-3-snake-eater/?region=All. [Consulté le 01/03/2024]

Pikmin. 2001. GameCube. Plus de 1,6 million d'unités vendues. https://www.vgchartz.com/game/1739/pikmin/?region=All. [Consulté le 01/03/2024]

Resident Evil 4. 2005. GameCube. Plus de 1,6 million d'unités vendues. https://www.vgchartz.com/game/1884/resident-evil-4/?region=All. [Consulté le 01/03/2024]

Shenmue. 2000. Dreamcast. Plus de 1,2 million d'unités vendues. https://www.vgchartz.com/game/2031/shenmue/?region=All. [Consulté le 01/03/2024]

Silent Hill 2. 2001. PlayStation 2. Plus de 1,2 million d'unités vendues. https://www.vgchartz.com/game/12685/silent-hill-2/?region=All. [Consulté le 01/03/2024]

Sly Cooper and the Thievius Raccoonus. 2002. PlayStation 2. Plus de 1,2 million d'unités vendues. https://www.vgchartz.com/game/6421/sly-cooper-and-the-thievius-raccoonus/?region=All. [Consulté le 01/03/2024]

Sonic Heroes. 2004. GameCube. Plus de 3,1 million d'unités vendues. https://www.vgchartz.com/game/5707/sonic-heroes/?region=All. [Consulté le 01/03/2024]

Star Wars: Knights of the Old Republic. 2003. Xbox. Plus de 2,1 million d'unités vendues. https://www.vgchartz.com/game/6445/star-wars-knights-of-the-old-republic/?region=All. [Consulté le 01/03/2024]

Super Mario Sunshine. 2002. GameCube. Plus de 5,9 million d'unités vendues. https://www.vgchartz.com/game/2281/super-mario-sunshine/?region=All. [Consulté le 01/03/2024]

Super Smash Bros. Melee. 2001. GameCube. Plus de 7,4 million d'unités vendues. https://www.vgchartz.com/game/2291/super-smash-bros-melee/?region=All. [Consulté le 01/03/2024]

Tekken 5. 2005. PlayStation 2. Plus de 2,8 million d'unités vendues. https://www.vgchartz.com/game/2346/tekken-5/?region=All. [Consulté le 01/03/2024]

The Legend of Zelda: Wind Waker. 2003. GameCube. Plus de 4,4 million d'unités vendues. https://www.vgchartz.com/game/2400/the-legend-of-zelda-the-wind-waker/?region=All. [Consulté le 01/03/2024]

The Lord of the Rings: The Two Towers. 2002. PlayStation 2. Plus de 4,6 million d'unités vendues. https://www.vgchartz.com/game/6318/the-lord-of-the-rings-the-two-towers/?region=All. [Consulté le 01/03/2024]

The Simpsons: Hit and Run. 2003. PlayStation 2. Plus de 4,7 million d'unités vendues. https://www.vgchartz.com/game/6419/the-simpsons-hit-amp-run/?region=All. [Consulté le 01/03/2024]

The Sims: Bustin' Out. 2003. PlayStation 2. Plus de 2,4 million d'unités vendues. https://www.vgchartz.com/game/5852/the-sims-bustin-out/?region=All. [Consulté le 01/03/2024]

Tom Clancy's Splinter Cell. 2002. Xbox. Plus de 3,2 million d'unités vendues. https://www.vgchartz.com/game/2467/tom-clancys-splinter-cell/?region=All. [Consulté le 01/03/2024]

Tomb Raider: Angel of Darkness. 2003. PlayStation 2. Plus de 1,2 million d'unités vendues. https://www.vgchartz.com/game/2475/lara-croft-tomb-raider-the-angel-of-darkness/?region=All. [Consulté le 01/03/2024]

Tony Hawk's Underground. 2003. PlayStation 2. Plus de 3,9 million d'unités vendues. https://www.vgchartz.com/game/6513/tony-hawks-underground/?region=All. [Consulté le 01/03/2024]

7E GÉNÉRATION DE CONSOLES

Animal Crossing: City Folk. 2008. Wii. Plus de 4,3 million d'unités vendues. https://www.vgchartz.com/game/24660/animal-crossing-city-folk/?region=All. [Consulté le 01/03/2024]

Assassin's Creed IV: Black Flag. 2013. PlayStation 3. Plus de 3,9 million d'unités vendues. https://www.vgchartz.com/game/71597/assassins-creed-iv-black-flag/?region=All. [Consulté le 01/03/2024]

Bioshock Infinite. 2013. PlayStation 3. Plus de 1,8 million d'unités vendues. https://www.vgchartz.com/game/46736/bioshock-infinite/?region=All. [Consulté le 01/03/2024]

Call of Duty: Modern Warfare 2. 2009. PlayStation 3. Plus de 10 millions d'unités vendues. https://www.vgchartz.com/game/28847/call-of-duty-modern-warfare-2/?region=All. [Consulté le 01/03/2024]

Crackdown 2. 2010. Xbox 360. Plus de 1,1 million d'unités vendues. https://www.vgchartz.com/game/35021/crackdown-2/?region=All. [Consulté le 01/03/2024]

Donkey Kong Country Returns. 2010. Wii. Plus de 6,5 million d'unités vendues. https://www.vgchartz.com/game/45674/donkey-kong-country-returns/?region=All. [Consulté le 01/03/2024]

God of War III. 2010. PlayStation 3. Plus de 7,6 million d'unités vendues. https://www.vgchartz.com/game/12533/god-of-war-iii/?region=All. [Consulté le 01/03/2024]

Gran Turismo 6. 2013. PlayStation 3. Plus de 5,2 million d'unités vendues. https://www.vgchartz.com/game/72181/gran-turismo-6/?region=All. [Consulté le 01/03/2024]

Grand Theft Auto IV. 2008. PlayStation 3. Plus de 11 millions d'unités vendues. https://www.vgchartz.com/game/7103/grand-theft-auto-iv/?region=All. [Consulté le 01/03/2024]

Grand Theft Auto V. 2013. PlayStation 3. Plus de 20,3 millions d'unités vendues. https://www.vgchartz.com/game/65884/grand-theft-auto-v/?region=All. [Consulté le 01/03/2024]

Halo 4. 2012. Xbox 360. Plus de 9,9 millions d'unités vendues. https://www.vgchartz.com/game/51758/halo-4/?region=All. [Consulté le 01/03/2024]

Halo Wars. 2009. Xbox 360. Plus de 2,6 millions d'unités vendues. https://www.vgchartz.com/game/7715/halo-wars/?region=All. [Consulté le 01/03/2024]

Heavy Rain. 2010. PlayStation 3. Plus de 3 millions d'unités vendues. https://www.vgchartz.com/game/7703/heavy-rain/?region=All. [Consulté le 01/03/2024]

inFamous 2. 2011. PlayStation 3. Plus de 1,8 million d'unités vendues. https://www.vgchartz.com/game/39923/infamous-2/?region=All. [Consulté le 01/03/2024]

Just Cause 2. 2010. PlayStation 3. Plus de 1,7 million d'unités vendues. https://www.vgchartz.com/game/24231/just-cause-2/?region=All. [Consulté le 01/03/2024]

L. A. Noire. 2011. PlayStation 3. Plus de 3,2 millions d'unités vendues. https://www.vgchartz.com/game/33961/la-noire/?region=All. [Consulté le 01/03/2024]

Lightning Returns: Final Fantasy XIII. 2014. PlayStation 3. Plus de 1 million d'unités vendues. https://www.vgchartz.com/game/70939/lightning-returns-final-fantasy-xiii/?region=All. [Consulté le 01/03/2024]

LittleBigPlanet. 2008. PlayStation 3. Plus de 4,5 millions d'unités vendues. https://www.vgchartz.com/game/12390/littlebigplanet/?region=All. [Consulté le 01/03/2024]

Mario Kart Wii. 2008. Wii. Plus de 37,3 millions d'unités vendues. https://www.vgchartz.com/game/6968/mario-kart-wii/?region=All. [Consulté le 01/03/2024]

Mario Party 9. 2012. Wii. Plus de 3,1 millions d'unités vendues. https://www.vgchartz.com/game/51842/mario-party-9/?region=All. [Consulté le 01/03/2024]

Metal Gear Solid 4: Guns of the Patriots. 2008. PlayStation 3. Plus de 6 millions d'unités vendues. https://www.vgchartz.com/game/6970/metal-gear-solid-4-guns-of-the-patriots/?region=All. [Consulté le 01/03/2024]

MotorStorm. 2007. PlayStation 3. Plus de 3 millions d'unités vendues. https://www.vgchartz.com/game/3850/motorstorm/?region=All. [Consulté le 01/03/2024]

Super Mario Galaxy 2. 2010. Wii. Plus de 7,4 millions d'unités vendues. https://www.vgchartz.com/game/35084/super-mario-galaxy-2/?region=All. [Consulté le 01/03/2024]

Super Smash Bros. Brawl. 2008. Wii. Plus de 13,3 millions d'unités vendues. https://www.vgchartz.com/game/6962/super-smash-bros-brawl/?region=All. [Consulté le 01/03/2024]

Tekken 6. 2009. PlayStation 3. Plus de 3,1 millions d'unités vendues. https://www.vgchartz.com/game/13114/tekken-6/?region=All. [Consulté le 01/03/2024]

The Last of Us. 2013. PlayStation 3. Plus de 8,1 millions d'unités vendues. https://www.vgchartz.com/game/70350/the-last-of-us/?region=All. [Consulté le 01/03/2024]

The Legend of Zelda: Skyward Sword. 2011. Wii. Plus de 3,6 millions d'unités vendues. https://www.vgchartz.com/game/45669/the-legend-of-zelda-skyward-sword/?region=All. [Consulté le 01/03/2024]

The Sims 3. 2010. PlayStation 3. Plus de 2 millions d'unités vendues. https://www.vgchartz.com/game/43343/the-sims-3/?region=All. [Consulté le 01/03/2024]

Uncharted 3: Drake's Deception. 2011. PlayStation 3. Plus de 9,3 millions d'unités vendues. https://www.vgchartz.com/game/49054/uncharted-3-drakes-deception/?region=All. [Consulté le 01/03/2024]

Viva Piñata. 2006. Xbox 360. Plus d'un million d'unités vendues. https://www.vgchartz.com/game/2619/viva-pinata/?region=All. [Consulté le 01/03/2024]

8E GÉNÉRATION DE CONSOLES

Animal Crossing: New Horizons. 2020. Switch. Plus de 22,4 millions d'unités vendues. https://www.vgchartz.com/game/222779/animal-crossing-new-horizons/?region=All. [Consulté le 01/03/2024]

Assassin's Creed: Odyssey. 2018. PlayStation 4. Plus de 3,1 millions d'unités vendues. https://www.vgchartz.com/game/222151/assassins-creed-odyssey/?region=All. [Consulté le 01/03/2024]

Batman: Arkham Knight. 2015. PlayStation 4. Plus de 4,1 millions d'unités vendues. https://www.vgchartz.com/game/79437/batman-arkham-knight/?region=All. [Consulté le 01/03/2024]

Detroit: Become Human. 2018. PlayStation 4. Plus de 3,2 millions d'unités vendues. https://www.vgchartz.com/game/86175/detroit-become-human/?region=All. [Consulté le 01/03/2024]

Final Fantasy XV. 2016. PlayStation 4. Plus de 5 millions d'unités vendues. https://www.vgchartz.com/game/73124/final-fantasy-xv/?region=All. [Consulté le 01/03/2024]

Ghost of Tsushima. 2020. PlayStation 4. Plus de 2,4 millions d'unités vendues. https://www.vgchartz.com/game/221791/ghost-of-tsushima/?region=All. [Consulté le 01/03/2024]

God of War. 2018. PlayStation 4. Plus de 11 millions d'unités vendues. https://www.vgchartz.com/game/120125/god-of-war-2018/?region=All. [Consulté le 01/03/2024]

Halo 5: Guardians. 2015. Xbox One. Plus de 5 millions d'unités vendues. https://www.vgchartz.com/game/82148/halo-5-guardians/?region=All. [Consulté le 01/03/2024]

Horizon: Zero Dawn. 2017. PlayStation 4. Plus de 10 millions d'unités vendues. https://www.vgchartz.com/game/85674/horizon-zero-dawn/?region=All. [Consulté le 01/03/2024]

Luigi's Mansion 3. 2019. Switch. Plus de 6,3 millions d'unités vendues. https://www.vgchartz.com/game/222771/luigis-mansion-3/?region=All. [Consulté le 01/03/2024]

Mafia III. 2016. PlayStation 4. Plus de 2,8 millions d'unités vendues. https://www.vgchartz.com/game/85862/mafia-iii/?region=All. [Consulté le 01/03/2024]

Mario Kart 8 Deluxe. 2017. Switch. Plus de 26,7 millions d'unités vendues. https://www.vgchartz.com/game/153114/mario-kart-8-deluxe/?region=All. [Consulté le 01/03/2024]

Metal Gear Solid V Phantom Pain. 2015. PlayStation 4. Plus de 3 millions d'unités vendues. https://www.vgchartz.com/game/73202/metal-gear-solid-v-the-phantom-pain/?region=All. [Consulté le 01/03/2024]

NieR: Automata. 2017. PlayStation 4. Plus de 2,6 millions d'unités vendues. https://www.vgchartz.com/game/86399/nier-automata/?region=All. [Consulté le 01/03/2024]

No Man's Sky. 2016. PlayStation 4. Plus de 2,2 millions d'unités vendues. https://www.vgchartz.com/game/84663/no-mans-sky/?region=All. [Consulté le 01/03/2024]

Persona 5 Royal. 2019. PlayStation 4. Plus de 1,4 million d'unités vendues. https://www.vgchartz.com/game/224750/persona-5-royal/?region=All. [Consulté le 01/03/2024]

Pokémon Sword and Shield. 2019. Switch. Plus de 18,2 millions d'unités vendues. https://www.vgchartz.com/game/223748/pokemon-sword-shield/?region=All. [Consulté le 01/03/2024]

Red Dead Redemption II. 2018. PlayStation 4. Plus de 13,9 millions d'unités vendues. https://www.vgchartz.com/game/140683/red-dead-redemption-2/?region=All. [Consulté le 01/03/2024]

Shadow of the Tomb Raider. 2018. PlayStation 4. Plus de 1,7 million d'unités vendues. https://www.vgchartz.com/game/221357/shadow-of-the-tomb-raider/?region=All. [Consulté le 01/03/2024]

Spider-Man. 2018. PlayStation 4. Plus de 13,2 millions d'unités vendues. https://www.vgchartz.com/game/120133/marvels-spider-man/?region=All. [Consulté le 01/03/2024]

Super Mario Odyssey. 2017. Switch. Plus de 18 millions d'unités vendues. https://www.vgchartz.com/game/152831/super-mario-odyssey/?region=All. [Consulté le 01/03/2024]

Super Mario Party. 2018. Switch. Plus de 10,9 millions d'unités vendues. https://www.vgchartz.com/game/222153/super-mario-party/?region=All. [Consulté le 01/03/2024]

Super Smash Bros. Ultimate. 2018. Switch. Plus de 19,9 millions d'unités vendues. https://www.vgchartz.com/game/221289/super-smash-bros-ultimate/?region=All. [Consulté le 01/03/2024]

Tekken 7. 2017. PlayStation 4. Plus de 2,7 millions d'unités vendues. https://www.vgchartz.com/game/85260/tekken-7/?region=All. [Consulté le 01/03/2024]

The Legend of Zelda: Breath of the Wild. 2017. Switch. Plus de 18, 6 millions d'unités vendues. https://www.vgchartz.com/game/118753/the-legend-of-zelda-breath-of-the-wild/?region=All. [Consulté le 01/03/2024]

The Sims 4. 2017. PlayStation 4. Plus de 1,3 million d'unités vendues. https://www.vgchartz.com/game/192885/the-sims-4/?region=All. [Consulté le 01/03/2024]

The Witcher 3: Wild Hunt. 2015. PlayStation 4. Plus de 10, 8 millions d'unités vendues. https://www.vgchartz.com/game/71482/the-witcher-3-wild-hunt/?region=All. [Consulté le 01/03/2024]

Tom Clancy's The Division. 2016. PlayStation 4. Plus de 4,3 millions d'unités vendues. https://www.vgchartz.com/game/73095/tom-clancys-the-division/?region=All. [Consulté le 01/03/2024]

Uncharted 4: A Thief's End. 2016. PlayStation 4. Plus de 16,2 millions d'unités vendues. https://www.vgchartz.com/game/77189/uncharted-4-a-thiefs-end/?region=All. [Consulté le 01/03/2024]

Watch Dogs 2. 2016. PlayStation 4. Plus de 3,3 millions d'unités vendues. https://www.vgchartz.com/game/86722/watch-dogs-2/?region=All. [Consulté le 01/03/2024]

GLOSSAIRE

Les sources de ces définitions apparaissent dans le corps de la thèse, lors de la première occurrence et où l'entrée en question est explicitée.

AAA : un jeu AAA ou triple A est un jeu avec un budget extrêmement élevé. Par extension, on peut également parler d'un studio AAA lorsqu'ils produisent des jeux AAA, ou d'un moteur AAA lorsque celui-ci est utilisé pour développer des jeux AAA.

Application Programming Interface (API) : collection de bibliothèques pour le développement d'une application. Les API majoritaires sont DirectX et OpenGL.

Asset : un contenu produit pour le développement d'un jeu vidéo, par exemple des modèles 3D, des images, des musiques, des scripts et ainsi de suite.

Bibliothèques : en programmation, ensemble de fonctions, classes ou modules prêts à l'emploi et permettant de faciliter le développement de logiciels.

Blueprint : langage de *scripting* visuel présent dans le moteur de jeu *Unreal Engine*, disponible depuis la quatrième version. Celui-ci a remplacé le langage visuel *Kismet* du moteur *Unreal Development Kit*.

Bones **:** structure en 3D utilisé pour animer des entités et définir un squelette et des mouvements. Un squelette est composé de plusieurs *bones*, et est nommé *armature* dans le logiciel *Blender*.

Central Processing Unit (CPU) : processeur représentant un élément essentiel d'un ordinateur, en charge d'effectuer des calculs et de traiter des instructions indiquant aux autres composants ce qu'il faut faire. Il est souvent décrit, de manière imagée, comme étant le cerveau d'un ordinateur.

Code source : texte d'un programme informatique tel qu'initialement écrit par les développeurs.

Digital Content Creation (DCC) : logiciel permettant de créer un modèle 3D ou des illustrations en 2D, comme *Blender*, *3DS Max* ou *Maya*.

First-Person shooter (FPS) **:** style de jeu à la vue subjective dans lequel le joueur doit combattre des vagues d'ennemis avec une arme à feu.

Game Design : ce qui détermine chaque détail du fonctionnement des mécanismes interactifs du jeu (*gameplay*).

Game Design Document (GDD) : document qui présente les éléments qui seront inclus dans un jeu vidéo.

Gameplay : ensemble des mécanismes interactifs présents dans un jeu vidéo et déterminant la manière dont les joueurs interagissent. Le terme jouabilité est souvent utilisé comme synonyme.

Graphics Processing Unit (GPU) : micro-processeur spécialisé dans le traitement des données graphiques, utilisé dans les ordinateurs, consoles de jeux vidéo et autres appareils électroniques.

Kismet **:** ancien langage de *scripting* visuel présent dans le moteur de jeu public *Unreal Development Kit*, et utilisable exclusivement dans un niveau donné.

Linux Kit : outil de développement amateur de la console *PlayStation 2*.

Modding : processus de modification ou d'altération d'un jeu vidéo commercialisé, pouvant être des ajustements mineurs ou des révisions substantielles.

Monde ouvert : concept de jeux vidéo caractérisé par des environnements virtuels vastes, offrant aux joueurs une grande liberté pour explorer les lieux et choisir leurs propres objectifs.

Moteur de jeu : Expression apparue dès 1994 et inspirée du milieu automobile, utilisée dans l'industrie vidéoludique pour désigner un ensemble d'outils, de bibliothèques et de technologies plus ou moins réutilisables, conçus pour faciliter le développement d'un jeu vidéo.

Moteur graphique : outil ou module permettant de gérer l'affichage graphique d'un logiciel, notamment des modèles 3D, des particules ou des images en 2D. Il peut être indépendant, comme *Irrlicht* ou *OGRE*, ou être intégré dans un moteur de jeu complet, comme c'est le cas dans la totalité des moteurs de jeu.

Net Yaroze **:** version spécifique de la première *PlayStation*, permettant le développement amateur de jeux vidéo, et nécessitant l'utilisation du langage *C*.

Open Source : code source disponible et consultable par n'importe quel individu.

Polygon Mesh : parfois simplement appelé *mesh*, cela désigne une représentation tridimensionnelle d'un objet créé à partir de polygones, notamment des triangles ou quadrilatères.

Skeletal Mesh : dans le moteur *Unreal Engine*, désigne un *mesh* 3D utilisé pour représenter des entités animés avec une structure squelettique.

Static Mesh : dans le moteur *Unreal Engine*, appellation pour désigner des objets ou entités inanimés, par exemple des bâtiments, meubles ou trésors.

UnrealScript : ancien langage de script, qui était disponible dans les trois premières versions du moteur *Unreal Engine*. Il a été remplacé à la fois par le langage visuel *Blueprint*, mais également par le *C++*.

What You See Is What You Get (WYSIWYG) : environnement visuel d'un logiciel de développement, comme un moteur de jeu, permettant d'assembler les *assets* dans un niveau et prévisualiser directement le résultat final.

World Composition : outil spécifique au moteur Unreal Engine 4 permettant de créer et gérer des mondes virtuels de grande taille.

TABLE DES MATIÈRES

Sommaire _____ **II**

Pages liminaires _____ **IV**

Remerciements _____ V

Conventions de la thèse _____ VI

Liste des acronymes et sigles _____ XI

1. Introduction : Genèse de l'interactivité _____ **1**

1-1. Contextualisation multifacette _____ 2

1-1-A. Taxinomie des genres vidéoludiques _____ 16

1-1-B. Problématique polymorphique _____ 23

1-1-C. Corpus développemental et vidéoludique _____ 33

1-2. Scientifisation des jeux vidéo _____ 42

1-2-A. Incompatibilité des *Game Studies* _____ 50

1-2-B. Informatique et développement de jeux _____ 58

1-2-C. Sémiologie et sémiotiques _____ 66

1-3. Structure globale de la thèse _____ 74

2. Développement Logiciel Tridimensionnel : Spécifications _____ **82**

2-1. Passerelle technologique infrangible _____ 83

2-1-A. Fortification technologique _____ 89

2-1-B. Obsolescence inéluctable _____ 94

2-1-C. Véridiction technico-discursive _____ 100

2-2. Générateur de théories vidéoludiques _____ 107

2-2-A. Symbiose sémio-informatique _____ 110

2-2-B. Interthéoricité logicielle _____ 114

2-2-C. Hyperstructure tridimensionnelle _____ 116

2-3. Synthèse du DLT _____ 119

3. Cinquième génération : Prémices des moteurs 3D _____ **124**

3-1. Assemblage des premiers jeux 125
3-1-A. Développement primo-générationnel 132
3-1-B. Prototypage des premiers moteurs de jeu 141
3-1-C. Obfuscation du développement amateur 153
3-2. Sémiotisation interactive des univers 3D 157
3-2-A. Abstraction tensive 166
3-2-B. Interactant tridimensionnel 172
3-2-C. Prélude des mondes ouverts 178
3-3. Synthèse de la cinquième génération 184

4. Sixième génération : Incrémentation des moteurs 3D 188

4-1. Technologisation vidéoludique 189
4-1-A. Incorporation du C++ 195
4-1-B. Développement des moteurs de jeu AAA 201
4-1-C. Démocratisation des outils indépendants 210
4-2. Caractères primordiaux interactifs 214
4-2-A. Interactivité générationnelle 218
4-2-B. Degré d'agnosticisme développemental 223
4-2-C. Faisceaux et substrats virtuels 227
4-3. Synthèse de la sixième génération 233

5. Septième génération : Polymorphie des moteurs 3D 238

5-1. Super-ordinateurs de divertissement 239
5-1-A. Généralisation des moteurs 3D internes 242
5-1-B. Unification indépendante 245
5-1-C. Kit de développement protéiforme 248
5-2. Interactants technologiques 252
5-2-A. Vision développementale du monde virtuel 258
5-2-B. Niveau sémique du gameplay 263
5-2-C. Isotopie d'un monde ouvert 267
5-3. Synthèse de la septième génération 269

6. Huitième génération : Hyperstructure des moteurs 3D 272

6-1. Standardisation des technologies 273

6-1-A. Réutilisation des moteurs internes 276
6-1-B. Expansion des moteurs de jeu publics 278
6-1-C. Démocratisation du moteur Unreal Engine 281
6-2. Hyperstructure 3D 284
6-2-A. Dimension technologique 286
6-2-B. Dimension interactive 291
6-2-C. Dimension structurelle 294
6-3. Synthèse de la huitième génération 296

7. Conclusion : Immuabilité technologique 298

7-1. Technologisation multidimensionnelle 299
7-1-A. Prémices des moteurs tridimensionnels 301
7-1-B. Démocratisation des moteurs de jeu 304
7-1-C. Réunification de l'industrie 306
7-2. Standardisation industrielle protéiforme 307
7-2-A. Rétrospection scientifique et difficultés 308
7-2-B. Conception d'une approche scientifique 311
7-2-C. Développement d'une hyperstructure 3D 313
7-3. Infinité des univers quantiques 315

Références 318

Bibliographie 319
Ludographie 341
Médiagraphie 345
Liste des illustrations 356

Pages annexes 358

Annexe A : Schémas originaux 359
Annexe B : Nouveau corpus vidéoludique 360
Annexe C : Présence des genres vidéoludiques 362
Annexe D : Corpus développemental indirect 366
Annexe E : Ancien corpus vidéoludique 368

Glossaire 381

Table des matières_____ 384

Approche sémio-informatique du Développement Logiciel Tridimensionnel : Programmation, interthéoricité et interactivité vidéoludique

Résumé

Cette recherche doctorale de sémiotique appliquée développe une approche inédite, nommée Développement Logiciel Tridimensionnel, qui relie informatique et sémiologie en interdisciplinarité. Son objectif majeur est d'analyser l'existence, voire l'influence des genres vidéoludiques dans la création des mécanismes interactifs, autrement dit du *gameplay*, au sein des outils de développement de jeux vidéo en 3D temps réel contemporains. Cela de manière diachronique, à travers la segmentation temporelle des générations de consoles, plus précisément de la cinquième jusqu'à la huitième génération de consoles.

Mots-clés : informatique, sémiotique, logiciels, programmation, interthéoricité, interactivité

Semiotics and Computer Science Methodology for Three-Dimensional Software Development: Programming, Intertheoreticity, and Gaming Interactivity

Summary

This doctoral research in applied semiotics introduces a pioneering approach, named Three-Dimensional Software Development, that integrates semiotics and computer science in a truly interdisciplinary manner. Its main objective is to analyze the very existence and influence of video game genres in the creation process of interactive mechanisms, namely the gameplay, within contemporary real-time 3D video game development tools. This is done diachronically, through the temporal segmentation of console generations, specifically from the fifth to the eighth generation of consoles.

Keywords: computer science, semiotics, software, programming, intertheoreticity, interactivity

UNIVERSITÉ SORBONNE UNIVERSITÉ

ÉCOLE DOCTORALE :
ED 5 – Concepts et langages
Maison de la Recherche, 28 rue Serpente, 75006 Paris, FRANCE

DISCIPLINE : Linguistique