UNIVERSITÉ DU QUÉBEC À MONTRÉAL

# *GAME GENESIS*

# UN PROFIL UML POUR LA RÉDACTION DE *GAME DESIGN DOCUMENTS*

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

ELLEN HAAS

DÉCEMBRE 2019

# REMERCIEMENTS

Tout d'abord je souhaite remercier mon directeur de recherche, Guy Tremblay. Pour sa présence, son temps ainsi que sa patience tout au long de ma maîtrise. Je souhaite aussi le remercier pour la confiance qu'il m'a accordée lorsque je lui ai proposé un sujet de recherche de ma propre initiative. Ses conseils et son expérience m'ont accompagnée tout au long de ma recherche et de ma rédaction.

Je souhaite également remercier mon conjoint, Jehan. Il m'a encouragée dans les moments de travail intense, épaulée durant les moments compliqués et m'a secouée dans les moments où je ne pensais plus y arriver. Ses paroles rassurantes, ses encouragements bruyants et la confiance inébranlable qu'il a exprimée tout au long de cette maîtrise ont été pour moi le socle solide qui m'a permis de continuer.

Je souhaite également remercier des professeurs qui ont joué un rôle important dans ma vie. M. Flieg qui a su rallumer en moi l'envie d'étudier et de découvrir de nouvelles choses en appréhendant le monde sous un angle différent. M. Froeliger sans qui j'aurais abandonné l'idée de faire mon échange à l'UQAM avant même d'être venue.

Je remercie également mes amis. Qu'ils soient tout près ou très loin. Que je les côtoie toutes les semaines ou que je les voie rarement. Ils sont nombreux, je ne les citerai donc pas, mais ils se reconnaîtront.

Enfin je tiens à remercier ma famille. Mon père tout d'abord, sans qui rien n'aurait été possible. Pour la confiance qu'il m'accorde, pour la patience dont il a toujours fait preuve et pour son soutien au quotidien depuis de nombreuses années. Je

remercie également ma grande soeur qui, même si elle est loin, a toujours un oeil sur ce que je fais.

# DÉDICACE

Je dédie ce mémoire à mon fils, Nahel.
Pour que tu puisses avoir la même chance que moi
de faire tes propres choix dans la vie.
Pour que je puisse te soutenir et t'épauler
dans le chemin que tu suivras
comme ton grand-père l'a fait pour moi.

TABLE DES MATIÈRES

# LISTE DES FIGURES

# LISTE DES TABLEAUX

# RÉSUMÉ

Le développement de jeux vidéos est un domaine en pleine expansion. Les jeux deviennent plus complexes, les projets plus compliqués, les budgets plus importants, les délais de développement plus serrés. Afin d'accélérer les étapes du développement d'un jeu, de nombreux outils émergent : moteurs de jeu, environnements de développement intégrés, outils de gestion de projet, logiciels de création 3D, aide au développement d'intelligences artificielles, gestion des animations, etc. Cependant, un pan complet de la création de jeux vidéos reste encore peu formalisé et peu outillé : le *game design*. Pourtant, la réflexion sur le *game design* et la documentation des concepts d'un jeu vidéo sont des étapes cruciales. Sans une solide documentation et des concepts clairement énoncés, un projet de développement peut être rapidement voué à l'échec.

Dans le cadre de ce mémoire, nous avons cherché à identifier les bonnes pratiques de rédaction d'un *Game Design Document* ainsi que les outils utilisés pour les étapes de *game design*. Comme résultat de ce travail, nous proposons *Game Genesis*, un profil UML pour faciliter la rédaction d'un *Game Design Document*.

UML introduit un langage formel, une structure précise, des outils performants et un support de communication efficace. Le domaine du développement de jeux vidéos est cependant très vaste. La taille des équipes de développement, les genres de jeux vidéos, les types de mécaniques de jeu (*gameplay*), les spécificités de *game design* entraînent une difficulté à anticiper tous les éléments nécessaires à la rédaction d'un *Game Design Document*. La liste des éléments introduits dans *Game Genesis* est donc non exhaustive et ces éléments sont assez généraux afin de ne pas faire obstacle à la représentation de certains types de jeux vidéos.

Afin de structurer le profil *Game Genesis*, nous avons fait usage du *framework MDA*, qui sépare le *game design* en trois aspect : *Mechanics*, *Dynamics* et *Aesthetics*. Plus spécifiquement, avec *Game Genesis*, nous proposons un profil pour aider à modéliser les éléments de *Mechanics* d'un jeu vidéo. Nous illustrons l'utilisation de ce profil en modélisant les éléments de *Mechanics* de PUBG (*PlayerUnknown's Battlegrounds*), un jeu vidéo populaire sorti en 2017.

MOTS CLÉS : *Game design*, jeux vidéos, *Game Design Document*, *framework MDA*, profil UML.

# INTRODUCTION

Le développement de jeux vidéos est un domaine en pleine expansion. Les jeux deviennent plus complexes, les projets plus compliqués, les budgets plus importants, les délais de développement plus serrés. De nombreux défis entourent donc le développement de jeux vidéos et les outils et méthodes pour les relever sont variés : moteurs de jeu, environnements de développement intégrés, outils de gestion de projet, logiciels de création 3D, aide au développement d'intelligences artificielles, gestion des animations, etc.

Cependant, un pan complet de la création de jeux vidéos reste encore peu formalisé et peu outillé : le *game design*. Pourtant, la réflexion sur le *game design* et la documentation des concepts d'un jeu vidéo sont des étapes cruciales. Sans une solide documentation et des concepts formulés de façon claire, un projet de développement peut être rapidement voué à l'échec. En outre, un *game designer* se doit d'être précis et concis dans ses descriptions et ses communications afin de transmettre un maximum d'informations et, surtout, afin de pouvoir dédier plus de temps à l'exploration, à la recherche artistique et au développement d'un prototype.

Dans ce mémoire, nous présentons *Game Genesis* (GG), un profil UML — mécanisme qui permet d'étendre le langage UML afin de l'adapter à un domaine ou un environnement particulier — qui permet d'outiller le processus de conception d'un jeu vidéo, facilitant ainsi la documentation et la rédaction des documents descriptifs du jeu vidéo en cours d'élaboration.

Destiné à être utilisé dans les premières phases du développement, *Breakthrough*

et Conception, le profil *Game Genesis* permet de documenter, à l'aide d'une notation graphique, les éléments de **mécanique** d'un jeu — au sens du *framework Mechanics Dynamics Aesthetics* (MDA) [5] — c'est-à-dire, les principaux éléments du jeu et les association et interactions entre eux. En d'autres mots, l'utilisation de *Game Genesis* aide à documenter le *modèle conceptuel* d'un jeu, les principaux concepts qui le caractérisent.

Dans ses grandes lignes, ce mémoire est organisé comme suit. Le chapitre 1 présente les principaux concepts liés au développement de jeux vidéos, notamment la notion de mécanique de jeu. Nous y traitons également des étapes de création d'un jeu vidéo, des notions d'exploration et d'exploitation, des moteurs de jeux et des genres vidéo-ludiques.

Le chapitre 2 présente différentes approches proposées dans la littérature pour documenter un *game design*. Nous y présentons le contenu d'un *Concept Document*, d'un *five (ou ten) pager* et d'un *Game Design Document*.

Le chapitre 3 présente le *framework* MDA de Hunicke, Leblanc et Zubek [5], qui découpe un jeu en trois aspects distincts : les éléments de *Mechanics* (données et algorithmes), les éléments de *Dynamics* (comportement à l'exécution des éléments de *Mechanics*) et les éléments d'*Aesthetics* (émotions générées par l'expérience de jeu). Nous y discutons également de certaines limites du *framework MDA* et de quelques propositions alternatives.

Le chapitre 4 présente la notion de «profil» en UML, qui permet d'étendre les mécanismes d'UML afin de l'adapter à un domaine d'activité ou à une plateforme de développement particulière.

Le chapitre 5 présente *Game Genesis*, le profil UML que nous avons développé, pour aider à la modélisation des éléments de mécanique d'un jeu vidéo.

Finalement, le chapitre 6 présente un exemple d'application de *Game Genesis* en décrivant les éléments de *Mechanics* du jeu *PlayerUnknown's BattleGrounds*.

# CHAPITRE I

# LE DÉVELOPPEMENT DE JEUX VIDÉOS

Dans ce chapitre, nous présentons les concepts clés liés au développement de jeux vidéos, notamment, les étapes de création d'un jeu, la distinction entre exploitation et exploration, les moteurs de jeux, les types de mécaniques de jeu. Mais tout d'abord, nous présentons une notion qui revient souvent dans les sections qui suivent, soit celle de mécanique de jeu.

## 1.1 La notion de mécanique de jeu

La mécanique d'un jeu vidéo est la façon dont le joueur se servira des éléments de l'environnement pour progresser vers le but du jeu. Le joueur définit une stratégie et met en place les éléments comme il le souhaite afin de remplir les objectifs du jeu auquel il joue. Dans un MMORPG — *Massive Multiplayer Online Role-Playing Game*, un genre de jeu vidéo défini à la Section 1.5 — la mécanique de jeu pourrait donner lieu à un scénario comme le suivant :

— Je suis un guerrier ;

— Je vais devoir me battre contre un *boss*[1] effectuant des attaques de glace ;

— Ce *boss* est sensible aux attaques de feu ;

1. Dans un jeu vidéo, un *boss* est un type de créature puissante qui représente généralement un défi important, dont l'affrontement marque une étape décisive de l'histoire ou d'une zone.

— En toute logique, je dois équiper mon personnage en conséquence, donc je m'équipe d'une armure A me défendant contre la glace et d'une arme A faisant des dégâts de feu.

Rolling et Morris [14] avancent qu'une mécanique de jeu doit éviter d'être triviale afin de pouvoir laisser le choix à un joueur d'adapter sa stratégie en fonction de nombreuses caractéristiques. Cela peut amener à des choix qui semblent moins évidents mais qui deviennent plus efficaces, en fonction des éléments réunis dans le combat. Prenon le cas précédent de notre *boss* de glace.

— Le joueur sait que le *boss* a une seule attaque de glace dévastatrice ;

— Sur un équipement, disons Armure B, il a la possibilité d'annuler cette capacité du *boss* ;

— Cette Armure B possèdes des caractéristiques de protection contre la foudre et d'augmentation de dégâts.

Le joueur se retrouve alors devant deux choix qui peuvent tous deux sembler corrects :

1. Équiper l'Armure A afin de se protéger et limiter les dégâts du *boss* ;
2. Équiper l'Armure B, annuler l'attaque de glace du *boss*, tuer le *boss* plus rapidement avec l'augmentation de ses dégâts.

Les deux cas sont à prendre en compte par le joueur car ils ont chacun leurs avantages et leurs inconvénients. C'est ce type de choix, selon Rolling et Morris [14], qui apporte un *gameplay* riche et intéressant ; ils avancent même que l'équilibre entre plusieurs mécaniques de jeu est crucial pour l'intérêt d'un jeu.

Une option présente dans un jeu mais n'ayant aucun intérêt à être jouée est une erreur de *game design*, que Rolling et Morris qualifient de « stratégie dominée ».

---

Un affrontement avec un *boss* génère plus d'expérience et permet de ramasser des objets plus rares qu'un affrontement avec une créature normale.

Dans la même optique, une option présente qui semble être la seule viable ou supérieure à toutes les autres, peu importe les facteurs extérieurs, est qualifiée de « stratégie dominante ».

Ainsi, un type de mécanique de jeu réellement plus puissant que les autres enlèverait une partie du « *fun* » ressenti par le joueur, car ce ne serait plus une question de choix, mais juste une question de calculs. Un joueur souhaitant faire le choix d'une stratégie moins évidente et moins évidente serait désavantagé par une mécanique de jeu trop forte par rapport aux autres. C'est ce que Rolling et Morris [14] qualifient de « problème de la stratégie dominante ». Afin de contrer ce problème, ils avancent que laisser une marge de manoeuvre plus grande aux joueurs permet d'éviter ce souci en leur permettant de construire leurs propres stratégies. Ils considèrent donc que « Un jeu bien conçu ne devrait pas contenir d'options qui ne valent pas la peine d'être jouées » [14].

Par contre, ces mêmes auteurs précisent qu'une option de mécanique de jeu n'est pas un simple choix logique. Une mécanique de jeu doit être réfléchie et composée de nombreuses règles afin de pouvoir permettre la présence de plusieurs stratégies viables dans un même cas donné. Chaque choix doit avoir ses avantages et ses inconvénients propres afin que le joueur puisse faire un choix en fonction de ses connaissances du jeu et des connaissances de son environnement. Ce choix, s'il comprend plusieurs solutions viables à plusieurs niveaux, est alors un choix *non trivial.* Cette liberté de choix crée une sensation de plaisir pour le joueur. La réussite de la stratégie choisie par le joueur entraîne alors une satisfaction pour ce joueur. L'échec, quant à lui, crée une frustration qui entraîne le joueur vers une nouvelle phase de réflexion pour améliorer sa stratégie.

## 1.2 Les étapes de création d'un jeu vidéo

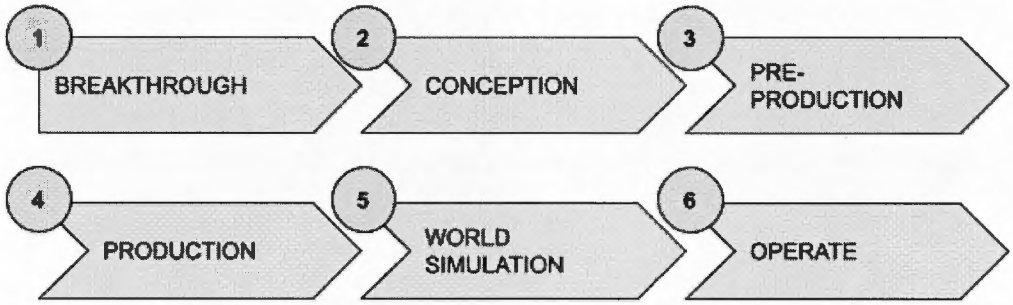


Figure 1.1: Les étapes de création d'un jeu vidéo.

Les étapes de développement d'un jeu vidéo ne sont pas immuables et dépendent de l'entreprise, du domaine d'activité ou des collaborateurs impliqués dans le projet. La figure 1.1 présente une liste non exhaustive de ces étapes, telle que présentée par Mathieu Nayrolles, architecte logiciel pour Ubisoft, lors d'un séminaire au LATECE (Laboratoire de recherche sur les Technologies du Commerce Électronique) de l'UQAM, le 10 avril 2019.

### 1.2.1 *Breakthrough*

Une étape de *Breakthrough* permet de réunir une équipe afin d'effectuer des recherches et des explorations sur des nouvelles mécaniques ou des nouvelles technologies afin de créer du nouveau contenu. Cette étape est optionnelle dans un projet. Deux exemples :

— Percée technologique, par ex., Google lance Stavia, une plateforme en ligne de jeux vidéos sous forme de catalogue et jouable à 100% en ligne, sans installation en local ;

— Percée de mécanique de jeu, par ex., génèse du mode *Battle Royale* (sect. 6.1).

### 1.2.2 Conception

Un document de conception est défini afin d'identifier l'environnement, la faisabilité et l'intérêt commercial du jeu. C'est durant cette étape que les *game designers* définissent et précisent l'univers, les mécaniques et le déroulement du jeu vidéo en question. Cette étape est majoritairement gérée par les *game designers* appuyés par les équipes des autres corps de métier présents dans un projet de développement de jeux vidéos, tels que les directeurs artistiques, les scénaristes, les modeleurs 3D, les graphistes, les développeurs, etc. Dans des projets à financements externes, cette étape est cruciale : elle permettra de présenter le projet à un studio avec une première maquette présentant les fondements du jeu.

### 1.2.3 Pré-production

Durant cette étape, des prototypes sont développés afin de créer une version minimale du jeu. Ces prototypes permettent d'avoir un aperçu jouable des concepts définis durant la phase précédente.

Un prototype est une coupe verticale de tout le système qui permet de valider ou redéfinir les concepts. Une fois le design bien défini, le prototype plus proche de ce que donnerait le jeu final et la faisabilité du projet confirmée, il est alors possible de rechercher les financements et les ressources nécessaires à la production. Cette étape implique tous les corps de métier dans un studio de développement, tous les aspects du jeu devant être représentés afin de montrer tout le potentiel du prototype.

### 1.2.4 Production

Une fois les fonds levés et les ressources humaines attribuées au projet, il est possible de procéder à la production du jeu vidéo. Tous les corps de métier sont représentés et le jeu est développé sous tous ses aspects et dans son intégralité. La plupart du temps le développement est découpé en plusieurs itérations, chacune permettant de vérifier que le jeu respecte bien les concepts définis plus tôt. C'est également à ce moment que les dernières modifications sont apportées aux concepts afin qu'ils respectent la vision du *game designer* et donc génèrent la bonne expérience. Dans le cas où le projet rencontre des contraintes supplémentaires (temps, argent, plateforme, etc.), les concepts peuvent également être revus. Généralement, durant cette étape, la publicité autour du jeu commence à prendre place afin d'informer le public et afin d'estimer l'impact que pourra avoir le jeu.

### 1.2.5 *World simulation*

Lors de la *World simulation*, tous les éléments du jeu sont testés et passés au crible. On vérifie que les éléments de jeu sont correctement modélisés, que les sons correspondent bien aux éléments, que les personnages correspondent à ceux décrits dans les documents de conception, etc. Plusieurs questions se posent à cette étape :

- Est-ce que les éléments interagissent bien entre eux ?
- Est-ce que l'univers de jeu est cohérent de bout en bout ?
- Est-ce que les mécaniques de jeu sont fluides et intuitives ?
- Est-ce que l'environnement de jeu est réellement tel que décrit par le *game designer* ?
- Est-ce que la bande sonore et la modélisation graphique génèrent bien les émotions attendues chez le joueur ?

### 1.2.6 *Operate*

Le jeu est maintenant produit et commercialisé. Une quantité importante de données est alors générée. Des bogues peuvent être remontés et corrigés pour des cas de figure particuliers ou inédits non couverts à l'étape de *World simulation.* Des ajustements mineurs peuvent être faits en fonction des besoins ou des demandes des joueurs. Le jeu prend alors toute sa dimension et toute sa vie à travers les joueurs.

Une fois le jeu bien en place et les étapes de corrections et ajustements passées, il est possible d'intégrer du nouveau contenu au jeu en repassant par les étapes précédentes. Ce nouveau contenu est généralement intégré au jeu sous forme de mises à jours, gratuites, ou de *Downloadable Content* (DLC), payants.

## 1.3 L'exploitation vs. l'exploration

### 1.3.1 L'exploitation

*L'exploitation* consiste à produire une suite à un jeu ou un nouveau jeu en utilisant des technologies (moteur, plateformes, etc.) ou une mécanique de jeu déjà existante mais avec du contenu additionnel. Ceci peut être fait dans le but de fidéliser une clientèle déjà existante — en ajoutant du contenu additionnel à un jeu existant —, d'offrir une expérience similaire avec des technologie plus récentes (par ex., FIFA) ou d'offrir une suite à un jeu ayant connu du succès (par ex., Dark Souls).

L'exploitation est une part importante du travail d'un studio de développement. De nombreux jeux vidéos récents sont fondés sur de l'exploitation de jeux précédents, autant au niveau des mécaniques de jeu qu'au niveau des concepts fondamentaux qui sont réutilisés de jeux précédents. C'est le cas des (grosses) pro-

ductions de franchises comme les jeux de EA sports (FIFA, NHL, NBA Live, Madden), les jeux d'action *role-play* de FromSoftware (série des Dark Souls), les jeux d'action aventure de Rockstar (série des GTA) ou les jeux de simulation de Maxi/EA Games (série des Sims).

### 1.3.2 L'exploration

*L'exploration*, ou l'innovation, dans le monde du jeu vidéo est essentielle au développement de nouveaux concepts de mécaniques de jeu mais également au développement de nouvelles technologies. La nouveauté est un enjeu essentiel afin d'attirer toujours plus de joueurs. L'investissement dans l'exploration est donc important pour un studio de développement. De la recherche de nouveaux concepts de jeux, de nouveaux types de mécaniques de jeu, de nouvelles technologies à intégrer ou de la création de nouveaux moteurs de jeux, l'exploration est devenue un facteur essentiel au domaine du jeu vidéo et à son expansion.

### 1.3.3 L'équilibre entre exploitation et exploration

Dans leur article, Parmentier *et al.* [11] explorent la capacité des studios à concilier ces deux activités. Ils présentent les enjeux de chacune d'entre elles et leur importance dans le domaine.

Il est nécessaire pour les studios de développement de jeux vidéos de trouver le juste équilibre entre exploitation et exploration. L'exploitation est le développement d'un jeu sur des mécanismes déjà existants où les règles sont déjà établies par un autre jeu ou un opus précédent. L'exploration est la découverte de nouvelles mécaniques de jeu ou la création de nouveaux outils de développement de jeux (comme un moteur de jeu). Le but est d'offrir aux clients des articles de qualité et attractifs. Cet équilibre est précaire et il est difficile pour un studio de

développement d'investir dans les deux domaines à la fois.

## 1.4 Les moteurs de jeux

Un moteur de jeu (*game engine*) est, typiquement, une suite logicielle contenant un *framework* de mécaniques de jeu — voir Section 3.2 — permettant d'accélérer le développement d'un jeu vidéo. Un moteur de jeu peut inclure une ou plusieurs facettes du développement du jeu : physique, graphismes, sons, calculs, gestion des périphériques d'entrée/sortie, gestion automatique de l'intelligence artificielle.

Voici une liste de certains des moteurs de jeu les plus connus accompagnés des jeux qui en font usage :

— *Unreal Engine* (`https://www.unrealengine.com/`) développé par Epic Games : Fortnite, Outlast 2, Dragon Ball Fighter Z, Days Gone.

— *Unity Engine* (`https://unity.com/`) développé par Unity Technologies : 7 Days to Die, Cuphead, Ori and the Blind Forest, Pokemon Go.

— *CryEngine* (`https://www.cryengine.com/`) développé par Crytek : Far Cry, Crysis 3, Deceit, Mavericks.

— *Frostbite* (`https://www.ea.com/frostbite`) développé par Dice (EA) : Battlefield V, Anthem, FIFA, Need for Speed.

Chaque moteur de jeu présente des avantages et des inconvénients en fonction du type de jeu que l'on souhaite développer. Par exemple, certains moteurs sont axés sur un type de jeu ou une plateforme en particulier, et ce afin d'être plus efficaces. L'innovation dans les moteurs de jeu est aussi essentielle au développement de nouveaux jeux. Par exemple, un moteur de jeu plus récent pourra intégrer des éléments comme des graphismes plus réalistes ou détaillés, ou des intelligences artificielles plus évoluées.

## 1.5 Les genres dans le jeu vidéo

L'exploration peut également consister en la création d'un nouveau type de mécaniques de jeu. Ce genre d'innovation est plus facilement identifiable par les joueurs et plus marquant en ce qui a trait à l'expérience de jeu. Ainsi, lors de la création d'un jeu vidéo, plusieurs mécaniques de jeu peuvent être sélectionnées afin de générer l'expérience voulue par le *game designer*. L'ensemble de ces mécaniques, réunies en un seul jeu, définissent le genre de ce jeu. Par exemple, dans le cas d'un *Battle Royale* (défini dans la liste ci-dessous), on allie les mécaniques de jeu de *Survival* et de *Last Man Standing*[2] afin de créer le genre *Battle Royale*.

Voici une liste non exhaustive des principaux genres présents dans les jeux vidéos : (`https://en.wikipedia.org/wiki/List_of_video_game_genres`)

- *MMORPG* (*Massive Multiplayer Online Role-Playing Game*) : un jeu en ligne massivement multijoueur mettant en scène un jeu de rôle avec différents objectifs à remplir : évolution par prise de niveaux, histoire principale vs. secondaire, développement social pour atteindre des objectifs sous forme de guilde, etc. (ex. : World of WarCraft, Black Desert Online).
- *Survival* : le joueur doit survivre à divers événements survenant dans le jeu. Il peut devoir subvenir à des besoins vitaux, construire de nouveau objets, survivre aux autres joueurs présents, etc. (ex. : Rust, Ark).
- Plateformes : un joueur contrôle un personnage qui se déplace dans un environnement de plateformes et doit progresser tout au long du niveau pour le compléter avant de pouvoir passer à un autre niveau (ex. : Mario, Donkey Kong).
- Simulation de vie : le joueur simule un environnement de vie plus ou moins

2. *Last Man Standing* : Jeu dans lequel la condition de victoire est d'être le dernier survivant.

réaliste et complexe en fonction des objectifs du jeu. La simulation peut s'appliquer à un personnage, à une ville entière, etc. (ex. : Les Sims, SimCity).

— FPS (*First-Person Shooter*) : le joueur, seul ou en équipe, doit se battre contre des ennemis (IA ou autres joueurs) à l'aide d'armes et d'équipements de combat (ex. : Call of Duty, Halo).

— *Beat-em up* : le joueur fait face à des vagues d'ennemis toujours plus forts (ex. : Bayonnetta, God of War).

— RTS (*Real-Time Strategy*) : des joueurs se font face dans un jeu où la gestion d'économie, de troupes et de populations est omniprésente afin de battre les autres joueurs (Age of Empire, Starcraft).

— 4X (*eXplore, eXpand, eXploit, and eXterminate*) : proche du RTS, ce genre est fondé sur une gestion pointue de ressources et de populations afin de battre les autres joueurs sous différents aspects et avec différents objectifs de victoire (population maximale, évolution de la société, critères financiers, etc.) (ex. : Civilization, Stellaris).

— MOBA (*Multiplayer Online Battle Arena*) : un type de mécanique de jeu où le jeu d'action rencontre le RTS. Plusieurs équipes de joueurs sont téléportées sur un territoire, chaque joueur contrôle un personnage et les joueurs doivent détruire la base de l'équipe adverse (ex. : League of Legends, DOTA).

— *Battle Royale* : plusieurs dizaines de joueurs sont parachutés sur un territoire où ils trouvent des armes et doivent s'entretuer ; le dernier survivant est déclaré vainqueur (ex. : Fortnite, PUBG/PlayerUnknown's BattleGrounds).

Les genres de jeux vidéos évoluent beaucoup. De nouveaux genres apparaissent grâce aux recherches exploratoires. Certains modes de jeux à succès deviennent des catégories à part entière. Il est aussi possible de combiner plusieurs modes de jeu afin de créer une nouvelle expérience. Ces divers genres de jeux vidéos

sont classifiés en fonction du type de monde, des objectifs de jeu, des actions nécessaires, etc.

Haitz et Law [4] ont mis en place une cartographie des genres afin de classifier les différents types de jeux en se référant à des caractéristiques précises des jeux et de leurs mécaniques. Cependant, il est difficile d'arriver à classifier tous les jeux tellement les genres sont nombreux et entrecoupés. C'est pour cela que la plupart des jeux sont classifiés dans des catégories larges et sont ensuite différenciés par leurs caractéristiques.

# CHAPITRE II

# LES DOCUMENTS DE *GAME DESIGN*

Dans ce chapitre, nous présentons plusieurs documents de *Game design* — *Game Design Document* (GDD) — qui peuvent être utilisés dans un processus de développement de jeux vidéos. Ces documents peuvent être rédigés l'un à la suite de l'autre, ou indépendamment l'un de l'autre — et ils ne sont pas obligatoires ou nécessaires dans chaque projet de jeu vidéo.

## 2.1 Des formes variées de documents de *game design*

Les documents de *game design* sont nombreux et leurs formes ainsi que leurs contenus sont souvent discutés dans la littérature. Selon le niveau d'avancement du développement, certains documents sont plus utiles que d'autres mais la plupart des acteurs du domaine sont d'accord pour avancer qu'il n'existe pas de gabarit fixe pour un document de *Game design* [6]. En fait, l'utilisation d'un gabarit trop spécifique et précis serait une erreur à éviter, car cela pourrait entraîner des difficultés de rédaction et d'adaptation face aux divers types de jeux que l'on peut rencontrer. Cependant, certains auteurs présentent des lignes directrices, un peu comme une recette de cuisine [13], qui peuvent être suivies ou dont on peut s'inspirer afin que le GDD soit complet et respecte une certaines forme.

Librande [7] montre des exemples de livrables de *game design* de différentes sortes.

Il souligne que les documents de design lourds sont des sources d'informations importantes réunissant tout le design dans un seul document. La création d'un document aide grandement à designer le jeu par la suite. Cependant, ces documents sont compliqués à maintenir, à mettre à jour et à parcourir pour trouver une information précise.

Librande présente également les «*design wikis*» qui apportent de nombreux points positifs au design. Il est possible d'y avoir accès à partir de n'importe quel endroit tant qu'une connexion réseau est disponible. Les mises à jours sont simplifiées et la recherche l'est également : il devient alors possible de modifier des éléments directement au cours d'une réunion. Un *wiki* permet aussi la contribution de tous les membres de l'équipe de développement aux différents éléments car la modification est possible par plusieurs utilisateurs. Il est possible d'éditer les éléments par article et donc de ne pas avoir à prendre connaissance d'éléments non reliés au travail actuellement effectué. Il est facile de maintenir un historique des versions et des modifications pour justifier les modifications et les garder en mémoire. Cependant, un *wiki* requiert une maintenance constante et donc, souvent, une personne dédiée à sa maintenance. Les relations entre les éléments ne sont pas mises en avant et il est compliqué de mettre ensemble différentes sortes de représentation sous la forme de textes/images. Autres désavantages : les images doivent être traitées à l'extérieur du *wiki* avant d'y être réintégrées, et l'impression d'un article *wiki* n'est pas toujours adaptée à une lecture rapide des éléments.

## 2.2 Le «*concept document*»

L'idée générale d'un «*concept document*» est d'avoir un document concis décrivant *le concept général d'un jeu*, mais sans entrer dans les détails, en indiquant

uniquement les principales lignes directrices.

### 2.2.1 Le «*One-Page*» de Librande

Librande [7] estime que plus un document est long, moins les utilisateurs s'y réfèrent. Afin de répondre à cette problématique, il a proposé le *One-Page Design*, un document qui donne une vue d'ensemble du jeu. Ce document est destiné à l'équipe de développement et aux décideurs des studios de production. Il doit donc contenir les informations essentielles pour présenter le projet de jeu sous forme visuelle, et ce sur une seule page [13].

Voici les caractéristiques essentielles d'un *One-Page design* telles que décrites par Librande [7] :

- Un titre représentatif du jeu.
- Les dates des divers éléments — afin de garder un historique.
- Des espaces entre les informations — afin de séparer chaque sujet et faciliter la compréhension.
- Une illustration centrale — pour focaliser l'attention.
- Sous l'illustration, possiblement une description et des textes explicatifs.
- Des légendes autour de l'illustration — pour des détails supplémentaires, ces légendes pouvant être des illustrations, elles-mêmes accompagnées de notes.
- Des barres latérales — pour ajouter des *checklists*, des objectifs principaux ou des informations diverses.

Selon Librande, la taille des éléments est importante dans un *One-Page design* : les tailles indiquent l'importance de l'information. Si nécessaire, un *One-Page design* peut être étendu à la taille nécessaire pour contenir toute l'information, y compris sous forme d'un poster. Cependant, l'augmentation de taille de la page doit quand

même permettre d'assurer sa lisibilité — augmenter en taille ne doit pas conduire à intégrer trop d'informations, ce qui irait à l'encontre de la lisibilité.

### 2.2.2 Le «*One-Sheet*» de Rogers

Rogers présente le «*One-Sheet*» comme vue d'ensemble d'un jeu [13]. Il insiste sur le fait que ce document doit être intéressant, informatif et court. Voici les éléments que doit contenir le *One-Sheet* de Rogers [13] :

— Le titre du jeu.

— Les systèmes de jeu prévus.

— L'âge des joueurs visés.

— La classification ESRB — voir plus bas.

— Un résumé de l'histoire du jeu en se concentrant sur les mécaniques de jeu.

— Les modes des mécaniques de jeu.

— Les USP — voir plus bas.

— Les jeux compétiteurs — voir plus bas.

Le ESRB — *Entertainment Software Rating Board* (ESRB) — est un organisme d'autorégulation qui est à l'origine d'un système de notation et de règles de respect de la vie privée des jeux et logiciels. Son système de notation permet de définir à partir de points précis l'âge conseillé pour l'utilisation d'un jeu ou logiciel. C'est une notation que l'on retrouve systématiquement dans le domaine du jeu vidéo définissant le type de contenu présent dans le jeu et le public pour lequel est recommandé le contenu : eC (*Early Childhood*), E (*Everyone*), E10 (*Everyone* $10^+$), T (*Teen*), M (*Mature* $17^+$), AO (*Adults Only* $18^+$).

Les USP — *Unique Selling Points* (USP) — sont des arguments de vente représentant le jeu. Ils sont généralement indiqués à l'arrière de la boite de jeu ou sur le

descriptif du jeu dans le cas d'une vente dématérialisée. Ce sont des points précis et courts attirant la curiosité de l'acheteur sans être trop descriptifs.

Les jeux compétiteurs indiquent les concurrents actuels du jeu, c'est-à-dire, les jeux déjà présents sur le marché. Cette liste doit contenir des jeux connus ou connaissant un grand succès afin qu'ils soient représentatifs du type du jeu décrit par le *One-Sheet.* Cela permet alors de donner une bonne idée de ce que sera le jeu. Présenter une liste de jeux obscurs ou qui n'ont connu que peu de succès découragera les éditeurs qui liront le *One-Sheet.*

## 2.3 Des extensions du document d'une page vers un résumé

Une fois les premières étapes de *Game design* effectuées, le document *One-Page* ou *One-Sheet* va être étendu afin de produire un document plus important. Ce second document va préciser les concepts du jeu en cours de création et va permettre de communiquer un niveau de détails plus précis.

Pedersen propose le *Five-Pager* [12], un résumé du concept du jeu et une description du jeu à venir. Le *Five-Pager* comprend toutes les informations essentielles au cycle de vie du jeu sous ses aspects majeurs, tels que les mécanique de jeu, l'audience cible, le scénario, et les *features.* Il permet de présenter le jeu de manière plus poussée que le *One-Page* à un décideur ou à un éditeur.

Quant à Rogers [13], il propose un document plus *fourni* avec son *Ten-Pager.* Le contenu du document doit non seulement contenir des informations pour l'équipe de développement du jeu, mais également les informations nécessaires pour intéresser les éditeurs impliqués dans le projet. Rogers présente le contenu du *Ten-Pager* sous la forme indiquée dans le tableau 2.1.

— La première page contient ce que le *One-Page* décrivait auparavant, soit un aperçu concis des caractéristiques du jeu ainsi qu'un calendrier prévisionnel de développement.

— Les pages 2 à 8 contiennent les informations sur le jeu en lui même : l'histoire, les personnages, les mécaniques de jeu, l'expérience que le joueur va rencontrer.

— La page 9 contient ce qui accompagne le jeu mais de l'extérieur : des succès à remplir,[1] des secrets et des *easter-eggs.*[2]

— La page 10 est un regroupement d'informations pratiques pour les décideurs du projet ou les potentiels financeurs afin d'expliciter les moyens de monétiser le jeu et d'en faire un projet lucratif.

---

1. Selon la plateforme sur laquelle un jeu est publié, il peut être accompagné de « succès », qui sont des tâches à accomplir et qui donnent droit à un trophée ou à une récompense quelconque.

2. Un secret peut être un niveau ou un objet collectionnable, alors qu'un *easter-egg* est une blague glissée dans un jeu par un développeur ou par l'équipe de développement.

Tableau 2.1: Le contenu du *Ten-Pager* selon Rogers [13].

- Page 1 : Informations générales
  - Titre du jeu
  - Systèmes de jeu prévus
  - Âge des joueurs visés
  - Notation ESRB
  - Calendrier prévisionnel de sortie
- Page 2 : Histoire
  - Résumé de l'histoire du jeu permettant de poser les premiers jalons de manière succinte et générale
  - Résumé du déroulement du jeu permettant de situer les actions du joueur dans l'histoire, les défis rencontrés par celui-ci, comment se déroule la progression, la place des mécaniques de jeu dans l'histoire, les conditions de victoire, etc.
- Page 3 : Détails du personnage contrôlé par le joueur
  - Histoire du personnage — caractère, traits importants de son apparence
  - Mécanique de jeu particulière associée au personnage, ses mouvements, ses armes
  - Maquette des contrôles proposés au joueur, par ex., représentation des raccourcis claviers
- Page 4 : Mécaniques de jeu
  - Modèle de séparation de l'histoire (niveaux, chapitres, monde ouvert)
  - Scénarios particuliers (cinématique active)
  - Mise en avant des USP
  - Diagrammes et illustrations apportant des précisions
- Page 5
  - Images et description du monde du jeu
  - Découpage des zones de jeu
  - Liens entre les zones

Tableau 2.1: Le contenu du *Ten-Pager* selon Rogers [13] (suite).

- Page 6 : Expérience de jeu
  - Description des émotions et sensations que doit générer l'expérience de jeu
  - Description de l'interface de jeu et de la manière d'en faire usage
- Page 7 : Mécaniques de jeu
  - Mécaniques : Eléments avec lequel un joueur interagit pour effectuer des actions (par ex., levier pour ouvrir une porte)
  - Dangers : Eléments du monde pouvant tuer ou blesser le joueur mais sans IA (par ex., bloc de pierre qui tombe)
  - *Power-up* : Objets que le joueur récupère et utilise pour obtenir un avantage (par ex., champignon dans Mario)
  - Objets de collections : Objets que le joueur collecte mais qui n'influence par directement les mécaniques de jeu (par ex., monnaie)
- Page 8 : Ennemis
  - Description des ennemis rencontrés par le joueur et qui sont contrôlés par une IA
  - Description des *boss* rencontrés
- Page 9 : Multijoueur et bonus
  - Description des succès collectionnables
  - Description des secrets découvrables
  - Description des interactions si le jeu est multijoueur
- Page 10 : Monétisation
  - Description du système de monétisation du jeu (Gratuit, Gratuit avec une boutique en jeu, payant à l'achat, etc.)
  - Description des boutiques et de leur contenu

## 2.4 Le *Game Design Document*

Le *Game Design Document* (GDD) est un des documents les plus complets utilisés dans le cadre du développement d'un jeu vidéo. Dans la littérature, il est souvent qualifié de «Bible du design» [12] — bien que certains auteurs ne soient pas d'accord sur ce sujet [13].

### 2.4.1 Le GDD, un moyen de communication

Le GDD doit contenir, dans l'idéal, toute la vision du *game designer* sans laisser de doute sur ce que celui-ci veut représenter et souhaite voir dans le jeu. Peu importe le corps de métier de la personne lisant le GDD, celle-ci doit pouvoir se représenter au maximum le rendu final attendu. Le GDD doit donc pouvoir servir de moyen communication entre les différentes équipes entourant le développement du jeu, et ce à toutes les étapes, depuis la création de l'idée jusqu'au post-mortem du projet.

### 2.4.2 Une structure souple mais un contenu précis

Freeman, dans un article sur Gamasutra [3], établit une liste de 10 pratiques qu'il estime nécessaire de respecter pour produire un GDD de qualité :

- Décrire le corps du jeu, mais également son âme.
- Rendre le GDD lisible.
- Établir des priorité dans les tâches.
- Entrer au maximum dans les détails.
- Décrire précisément les idées compliquées des concepts, quitte à les démontrer, les illustrer ou en faire des mini prototypes.
- Décrire le «quoi» mais également le «comment».

— Proposer des alternatives de mise en œuvre.

— Donner une vie au GDD.

— Préciser la vision du *game designer* de façon suffisamment précise pour ne pas laisser d'éléments non décrits.

— Mettre à disposition le GDD dans de bonnes conditions, c'est-à-dire que le contenu du GDD doit être consistant du début à la fin, en respectant une structure cohérente et un contenu détaillé, et qu'il soit mis à jour régulièrement.

### 2.4.3 Un GDD complet mais pas surchargé

Le GDD doit être un outil pour les *game designers*, tout en leur donnant la liberté de créer de nouveaux jeux, de nouvelles mécaniques de jeu, de nouveaux concepts, donc sans les brider dans leur créativité. De nombreux travaux essaient de répondre au besoin de formalisation du GDD [1, 3, 9, 15] sans pour autant réussir à apporter une solution permettant d'englober tous les types de jeux, tous les types de métiers ou toutes les étapes de développement. Et de nombreux problèmes annexes se posent concernant l'universalité d'un modèle de GDD : Est-il complet ? Est-il assez précis ? Est-il assez souple ? Est-il utile à l'équipe ?

Prenons l'exemple d'un projet où le GDD est extrêmement complet, comportant tous les détails voulus par le *game designer*. Le GDD peut alors devenir tellement complet qu'il en devient imposant, lourd à parcourir, difficile à maintenir et à modifier. Il perdra alors toute l'efficacité de design qu'il aurait pu donner, pouvant devenir un handicap plutôt qu'une aide au développement [7].

Un GDD est l'atout majeur de la phase de production d'un jeu vidéo et pour qu'il soit utile durant toutes les phases, celui-ci doit être impérativement complet et sans ambiguité [8]. Cependant, il doit être assez souple afin de suivre le jeu dans

son développement et suivre les changements nombreux et spontanés qui peuvent survenir au cours du développement.

Freeman [3] écrit que le GDD doit non seulement décrire le corps du jeu vidéo mais son «âme». Selon lui, construire un GDD doit être un moyen de présenter le résultat attendu dans les moindres détails pour permettre aux équipes de travailler sur une idée précise. Faire appel à des outils graphiques plutôt qu'à des textes peut être un moyen d'apporter plus de précision à une idée. Mais le GDD doit être un moyen d'expliquer toutes les parties du jeu, le «quoi», mais également le «comment».

Cependant, décrire l'intégralité des détails dans le GDD peut mener à des dérives décrites par Rouse [6]. À vouloir apporter trop de détails dans un GDD, un *game designer* peut perdre énormément de temps, qui aurait pu être économisé en ayant plus de communication avec son équipe. Trop de détails peut également apporter trop de limitations aux corps de métier plus artistiques et brider leur créativité. Certaines parties du jeu peuvent également en obstruer d'autres : trop de détails dans la description d'une mécanique peut éclipser le fait qu'une autre mécanique n'est pas assez détaillée. Un GDD trop complet donne également l'impression que le projet est terminé et qu'il ne nécessite plus d'ajustements, ce qui peut nuire à l'évolution du jeu et aux modifications positives qui pourraient devoir lui être apportées.

## 2.5 Conclusion

Dans ce chapitre, nous avons vu trois types de documents de *Game design* :

- *One-Page* et *One-Sheet*, qui servent de premier aperçu du jeu.
- *Five-Pager* et *Ten-Pager*, qui permettent de présenter le jeu à une équipe de développement et de référencer les premières pistes de *Game design*.

— *Game Design Document*, qui permet de regrouper tous les éléments de design pour un jeu en cours de conception ou de développement.

Ces documents sont les plus répandus dans le monde du développement de jeux vidéos.

Selon le type de mécanique de jeu, la quantité d'informations ou la durée de vie du jeu, un GDD peut avoir une taille plus ou moins importante. La structure du GDD peut différer d'un *game designer* à l'autre en fonction des habitudes de travail avec son équipe ou son expérience. Le contenu du GDD peut aussi prendre des formes diverses : textes, graphiques, diagrammes, *mind mapps.*

C'est pour ces raisons qu'il est compliqué d'établir un gabarit précis pour la rédaction d'un GDD. Aucun standard n'existe afin de définir précisément l'architecture des documents de *game design.* La rédaction de ces documents se fait en suivant des bonnes pratiques décrites par des *game designers* expérimentés. Chaque *game designer* peut définir son propre gabarit en fonction de ses besoins et de ses habitudes. Toutefois, tout GDD bien structuré et complet se devra de modéliser les aspects clés d'un jeu. Dans le prochain chapitre, nous présentons le *framework* MDA, qui identifie ces aspects clés et qui structure ces éléments de *game design* en trois parties : *les règles, le système et le fun.*

# CHAPITRE III

## LE *FRAMEWORK «MECHANICS, DYNAMICS, AESTHETICS»* (MDA)

La conception de jeux est le processus de réflexion nécessaire à la création d'un jeu. C'est l'évolution que subit le jeu depuis une idée initiale jusqu'à sa réalisation finale. De nombreuses étapes permettent de modéliser et formaliser tous les aspects du jeu pour le mener à sa version jouable : la mise en place du cadre spatio-temporel dans lequel le joueur évoluera, les règles qu'il devra suivre pour avancer, les éléments graphiques qu'il rencontrera. Tout cela est défini lors de la conception.

Un jeu vidéo est cependant difficile à concevoir et, surtout, à documenter. Un logiciel classique vise généralement à répondre à un besoin précis identifié et demandé par les utilisateurs. Le logiciel est donc une réponse à une problématique et son utilisation est logique si le logiciel répond au besoin énoncé. Par contre, un jeu vidéo doit souvent créer «un besoin» chez un utilisateur, qui n'a pas de besoins liés à son utilisation. Il doit proposer un but au joueur, lui donner l'envie de continuer à l'utiliser et doit générer des émotions pour être apprécié, car il ne vise pas à résoudre une problématique.

Il est compliqué de définir une méthode précise pour décrire une émotion ainsi que les moyens à mettre en place pour parvenir à la générer. Le *framework Mechanics Dynamics Aesthetics* (MDA) de Hunicke, Leblanc et Zubek [5] tente de définir une structure pour accompagner un *game designer* durant les étapes de documentation

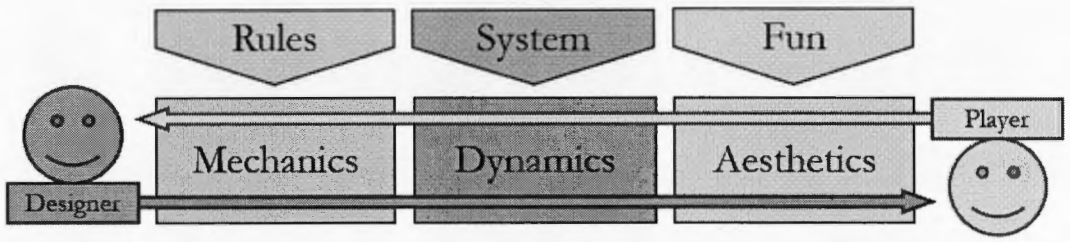


Figure 3.1: Les trois parties du *framework* MDA [5].

d'un jeu. MDA essaie d'apporter une solution en apportant une catégorisation des émotions et des approches et techniques pour parvenir à les générer.

## 3.1 Le jeu : un échange entre *game designer* et joueur

Un *game designer* crée un jeu dans le but de générer une expérience de jeu. Cependant, l'utilisation que le joueur fera du produit fini est difficile à prévoir, comme le décrivent Hunicke, LeBlanc et Zubek [5], qui découpent un jeu en trois aspects distincts : les règles, le système et le « *fun* ». Les flèches de la figure 3.1 montrent l'interaction entre ces trois aspects. Selon l'article de Hunicke, LeBlanc et Zubek [5], un *Game designer* met en place des *Mechanics* dont découlent des *Dynamics* dont naissent des *Aesthetics*. Le joueur quant à lui interagit avec des *Aesthetics*. Ces actions lui permettent de faire usage des *Dynamics* afin de déclencher les éléments de *Mechanics*. Ces trois aspects sont à l'origine des trois parties du *framework* MDA, tel qu'illustré dans la figure 3.1, que nous expliquons plus en détail dans les sections qui suivent.

## 3.2 *Mechanics*

L'aspect *Mechanics* d'un jeu est composé de tous les éléments du jeu associés à la représentation des données et des algorithmes. Les éléments de *Mechanics* sont

classés en diverses catégories, qui permettent de les définir plus précisément. Ces catégories peuvent comprendre des attributs et des spécificités qui sont appliquées aux divers éléments de chaque catégorie.

La *Mechanics* comprend également les actions, comportements et mécanismes de contrôles mis à disposition du joueur. Cela peut correspondre aux mouvements d'un personnage, aux actions possibles sur des objets ou aux interactions entre les objets. Les règles sont aussi définies dans les mécaniques.

## 3.3 *Dynamics*

Les éléments de l'aspect *Dynamics* représentent les conséquences des éléments de *Mechanics*. Ils décrivent le comportement de l'exécution de la *Mechanics*, lorsqu'ils seront effectivement utilisés par le joueur [2]. Ces éléments de *Dynamics* sont importants à prévoir car ce sont eux qui permettront l'évolution du joueur dans le jeu.

Afin de définir la *Dynamics*, il est possible d'analyser d'autres jeux (de même genre ou d'opus précédents) ; il est aussi possible d'effectuer des analyses statistiques sur les habitudes de jeu des joueurs, d'étudier leur psychologie afin de prévoir leur façon de jouer et manipulant leurs émotions.

## 3.4 *Aesthetics*

Les éléments de l'aspect *Aesthetics* sont, selon Hunicke, LeBlanc et Zubek [5], « ce qui rend le jeu *«fun»* ». Ce sont toutes les émotions générées par le jeu et transmises au joueur par l'intermédiaire de la mécanique de jeu, des sons ou des graphismes. Hunicke, Leblanc et Zubek classifient ces éléments en huit catégories :

— Sensation : le jeu comme plaisir des sens.

— Fantaisie : le jeu comme imaginaire.

— Narration : le jeu comme situation dramatique.

— Défi : le jeu comme parcours d'obstacles.

— Communauté : le jeu comme réseau social.

— Découverte : le jeu comme territoire inexploré.

— Expression : le jeu comme découverte de soi-même.

— Soumission : le jeu comme passe-temps.

Le jeu final peut contenir une ou plusieurs catégories de «fun» et l'ensemble de l'expérience du joueur repose sur ces éléments d'*Aesthetics*. Cette dernière partie du *game design* est si importante qu'il est possible de concevoir et développer le jeu en ayant au préalable sélectionné un certain nombre de ces catégories et en les considérant comme objectifs à atteindre.

## 3.5 Des évolutions du *framework* MDA

Le MDA permet de modéliser la conception du jeu vidéo en séparant trois aspects clés d'un jeu : les éléments de *Mechanics*, de *Dynamics* et d'*Aesthetics*. Cependant, de nombreux auteurs critiquent MDA car certains genres de jeu ou certaines mécaniques de jeu sont exclues de ce *framework*. D'autres *frameworks* ont donc été proposés. Par exemple, les *frameworks* « *Design, Play, Experience* » (DPE) (Sect. 3.5.1) — qui cherche à intégrer les jeux reposant uniquement sur l'histoire, les jeux sérieux ou les outils d'apprentissage ludique — et « *Design, Dynamics, Experience* » (DDE) (Sect. 3.5.2) — qui avance que MDA néglige certains aspects du design de jeux vidéos en se concentrant trop sur les éléments de *Mechanics*.

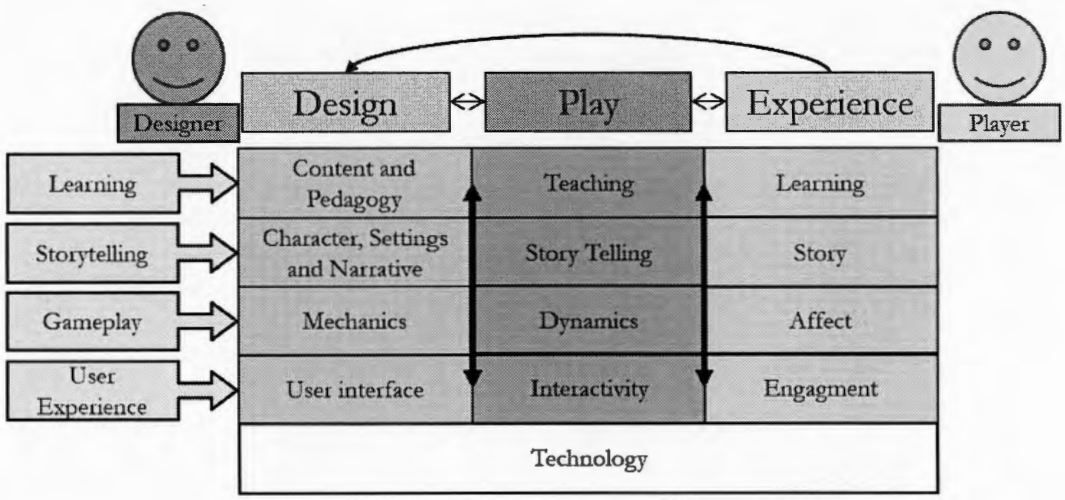


Figure 3.3: Les détails du *framework* DPE [21].

### 3.5.1 Le *framework* «*Design, Play, Experience*» (DPE)

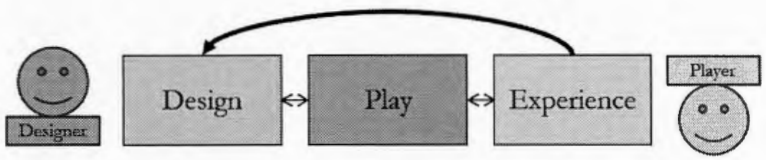


Figure 3.2: Les trois parties du *framework* DPE [21].

Le *Design Play Experience* (DPE), illustré à la figure 3.2, est un *framework* reposant sur les mêmes principes que MDA. Plus spécifiquement, des modifications sont appliquées au MDA afin d'étendre ses capacités de design pour les jeux sérieux. Ainsi, DPE étend MDA afin d'y intégrer les notions d'apprentissage, de scénarisation (*story telling*), de mécaniques de jeu et de composants technologiques spécifiques aux jeux sérieux. La figure 3.3 donne les différentes parties du DPE, telles que présentées par Winn [21].

Dans le *framework* DPE, le *game designer* a le contrôle direct sur l'ensemble des catégories. Notamment, le *game designer* définit explicitement des objectifs

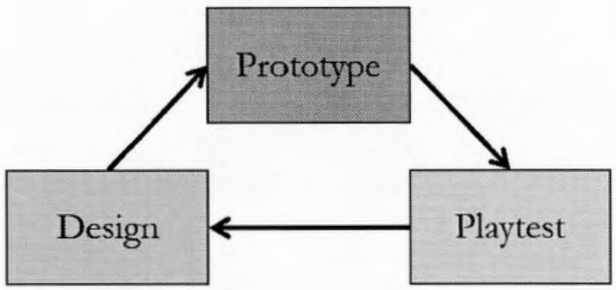


Figure 3.4: Le processus de design itératif associé à DPE [21].

d'*Experience*. Dans la figure 3.2 les flèches représentent les relations entre le *Game designer*, le jeu et le joueur. Le *Game designer* conçoit (*Design*) le jeu, le joueur joue (*Play*) au jeu, et il en résulte une expérience de jeu (*Experience*). La flèche allant d'*Experience* vers *Design* représente le processus itératif du design décrit par Salen et Zimmermann [17], tel qu'illustré à la figure 3.4. La conception permet de produire un prototype, et l'expérience sur ce prototype permet de modifier le design afin que l'*Experience* corresponde aux attentes du *game designer*.

### 3.5.2 Le *framework* « *Design, Dynamics, Experience* » (DDE)

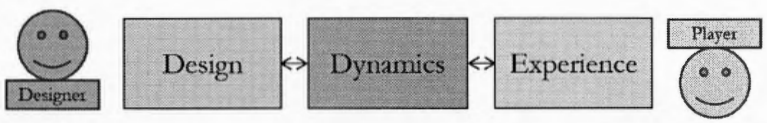


Figure 3.5: Les éléments du *framework* DDE [19].

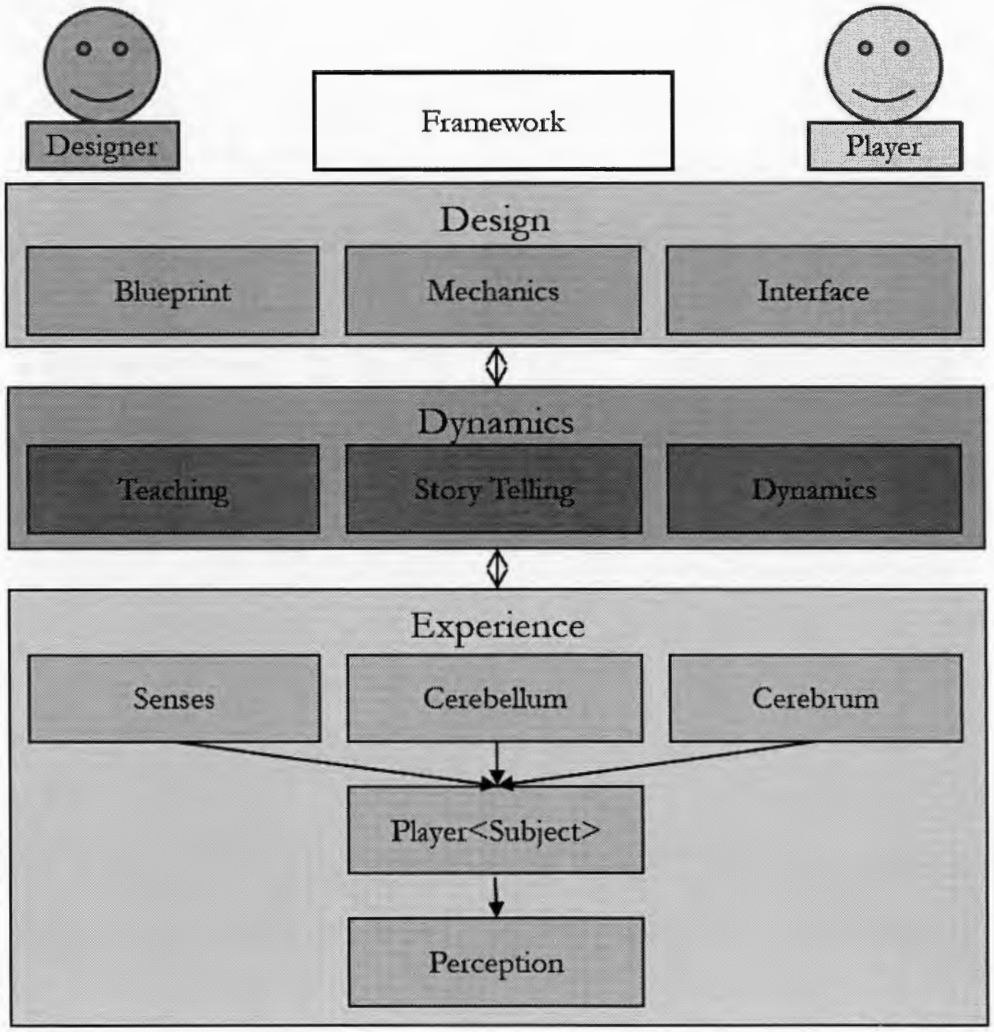


Figure 3.6: Les détails du *framework* DDE [19].

Le framework *Design Dynamics Experience* (DDE), illustré à la figure 3.5, a été proposé par Walk *et al.* [19]. Ils apportent une vision critique du *framework* MDA et avancent que celui-ci néglige de nombreux aspects du design de jeux vidéos. Ils estiment également que MDA se concentre trop sur l'aspect *Mechanics* et ne permet donc pas de décrire tous les types de mécaniques de jeu présents sur le marché. Ce focus sur l'aspect *Mechanics* entraîne, selon Walk *et al.*, un manque

de contrôle du *game designer* sur les aspect *Dynamics* et *Aesthetics* qui ne font que découler de l'aspect *Mechanics*.

Walk *et al.* effectuent donc un découpage différent et intègrent de nouvelles notions dans les catégories, telles qu'illustrées dans la figure 3.6. Dans cette figure, les flèches entre les trois parties indiquent que l'on garde l'interaction présente entre les trois parties que sont *Design*, *Dynamics* et *Experience*. Au sein de la section *Experience*, les flèches indiquent que la combinaison de *Senses*, *Cerebellum* et *Cerebrum* appliquée sur un joueur *en particulier* (*Player<Subject>*) devient ce que l'on appelle la *Perception*.

*Design*

Les éléments associés à l'aspect *Design* sont les suivants :

— *Blueprint* : Partie du design qui concerne les concepts du monde du jeu : culture, religion, physique, ensembles de règles, styles artistiques, design narratif, design de personnages et design sonore, qui ensemble créent l'expérience esthétique.

— *Mechanics* : Toute chose créant le jeu, plus précisément le code. L'architecture du code, la prise en charge des entrées/sorties, la prise en charge des objets, l'implémentation des règles de jeu et l'interaction entre les objets, et tous les éléments reliés au code. Cela comprend donc tous les éléments que le joueur ne perçoit pas directement dans son utilisation du jeu.

— Interface : Toutes les mécaniques qui ont pour but de communiquer le jeu au joueur. Les graphismes, les sons, les réactions et les interactions. Ces mécaniques peuvent aussi servir à effectuer une communication sous la forme Jeu–Joueur, Joueur–Jeu ou Jeu–Jeu. Cette partie comprend également les cinématiques, les textes affichés, et tout ce qu'il est possible de voir ou d'entendre dans le jeu.

*Dynamics*

La catégorie *Dynamics* de DDE correspond à celle de MDA, mais elle classifie les interactions de manière à les rendre plus précises, décomposées en trois sous-catégories :

— *Player–Game*

— *Player–Player*

— *Game–Game*

*Experience*

Dans le *framework* MDA, la troisième partie est l'*Aesthetics* : tout ce que le joueur sent et ressent lors de son activité et interaction avec le jeu. La partie *Experience* du jeu étend l'*Aesthetics* afin de prendre en considération que le joueur n'est pas uniquement l'ensemble des émotions générées par le jeu. Le joueur devient un élément avec une expérience déjà présente avant l'utilisation du jeu, ce qui peut modifier les émotions générées d'un joueur à l'autre. Une même couleur, un même son ou une même image peuvent générer différentes réactions de la part du joueur et la partie *Experience* de DDE essaie de prendre cela en compte.

— *Senses* : expérience sensorielle du joueur du début à la fin du jeu.

— *Cerebellum* : les émotions ressenties par le joueur.

— *Cerebrum* : les défis intellectuels et les décisions prises par le joueur.

— *Player<Subject>* : la partie que le designer ne peut pas contrôler. Il peut se baser sur les trois premières catégories afin de prévoir certaines réponses spécifiques du joueur. N'ayant aucun contrôle sur celles-ci, le designer ne peut qu'estimer la réaction du joueur en fonction d'objectifs et de logiques psychologiques pour l'amener à la réaction souhaitée.

— *Perception* : ce que ressent réellement le joueur en fonction des trois premières catégories et de sa propre expérience et personnalité en tant que joueur. Cela comprendra ses mécaniques de jeu, le type de défi qu'il perçoit, l'amusement qu'il ressent, la beauté qu'il perçoit, l'écho que génère l'histoire en lui, etc.

### 3.5.3 Une autre limite du *framework* MDA

Dans une critique du *framework* MDA, Duarte [2] met de l'avant les difficultés rencontrées à représenter, à l'aide de MDA, des jeux qui ne soient pas des jeux vidéos.

Duarte explique que les règles d'un jeu vidéo sont souvent implicites aux types de mécaniques de jeu ou genre de jeu vidéos. Cependant, dans le cadre des jeux de plateau, les règles ne peuvent pas être implicites et acquises par l'expérience. Elles doivent être explicites et explicables dans un manuel d'utilisation. Il fait l'analogie entre un jeu FPS — *First-Person Shooter* (FPS) — et un jeu d'échecs. Dans un FPS, un joueur sait à quoi s'attendre en fonction de son expérience du genre du jeu. Il saura où trouver les éléments et saura comment faire usage de l'interface graphique qui lui est présentée. Cependant, un joueur se trouvant devant un jeu d'échecs pour la première fois ne pourra pas acquérir par l'expérience les connaissances requises pour jouer : les règles devront lui être expliquées à la base et l'expérience pourra lui apporter des aspects tactiques ou stratégiques du jeu.

## 3.6 Conclusion

Nous avons donc vu dans ce chapitre que le *framework MDA* sépare la conception d'un jeu en trois aspects :

- *Mechanics* : Les **concepts** et les règles.
- *Dynamics* : Le système et son **comportement** dynamique.
- *Aesthetics* : Les **émotions** et le ***«fun»***.

Le but de cette recherche est d'apporter une aide à la rédaction d'un *Game Design Document* qui commence par représenter les éléments présents dans le jeu et par identifier les principales interactions entre eux. Dans ce mémoire, nous avons donc décidé de nous concentrer sur la modélisation des éléments de *Mechanics*, leur description ainsi que l'identification des interactions. C'est ce que nous présentons, dans le Chapitre 5, avec notre profil *Game Genesis*. Mais auparavant, nous devons introduire la notion de *profil UML*.

# CHAPITRE IV

# LES PROFILS UML

Un projet de développement de jeu vidéo est un projet de grande envergure. C'est non seulement un projet informatique, mais également un projet de design, un projet artistique visuel et sonore, et un projet de scénario.

Un projet de développement de jeu vidéo est complexe et c'est pour cela que les phases d'analyse et de conception sont essentielles à sa réussite. Il faut analyser les besoins et le domaine et organiser les étapes d'analyse et de conception. Toutes ces données doivent être accessibles et modifiables afin de pouvoir les consulter et les faire évoluer tout au long du projet.

Un outil permettant de modéliser les besoins, les données et la conception d'un logiciel existe déjà : *Unified Modeling Language* (UML). UML permet de modéliser tous ces aspects, mais peut également être adapté à un domaine d'application particulier à travers les *profils UML*. C'est ce que nous présentons dans le présent chapitre.

## 4.1 Le langage de modélisation UML

Le *Unified Modeling Language* (UML) est un langage de modélisation standardisé permettant de créer des modèles, typiquement sous forme de diagrammes.

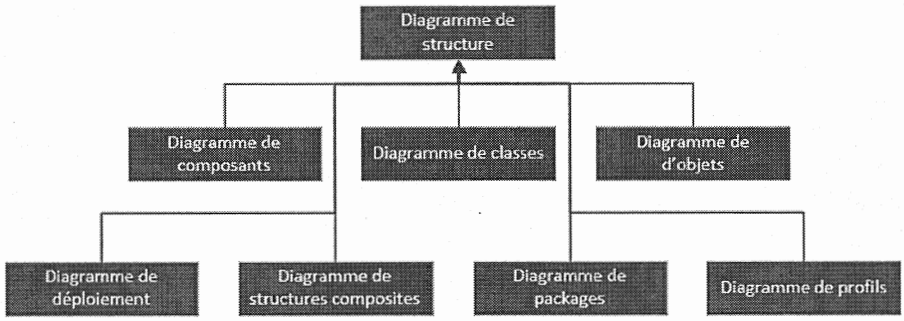


Figure 4.1: Les principaux types de diagrammes de structure d'UML [10].

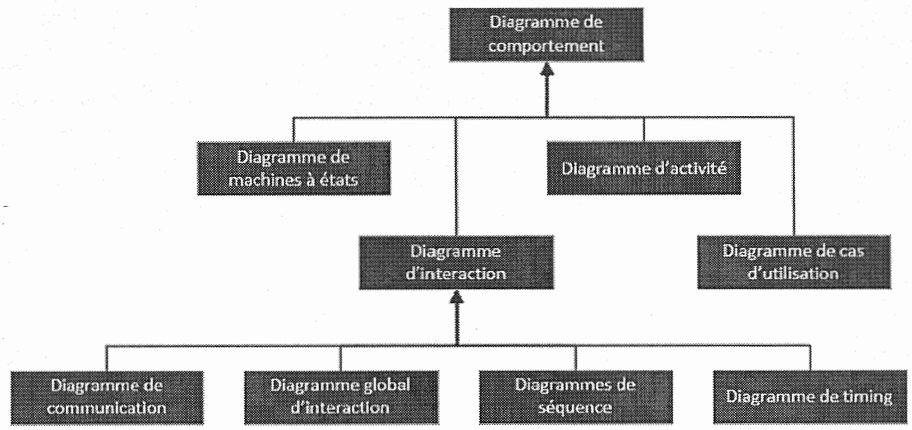


Figure 4.2: Les principaux types de diagrammes de comportement d'UML [10].

Ces diagrammes permettent de visualiser, spécifier, construire et documenter des logiciels, des systèmes et des processus d'affaire. UML utilise diverses notations, surtout graphiques, pour exprimer l'architecture, le design ainsi que la mise en œuvre de logiciels. Les spécifications du standard UML sont en ligne sur le site de l'OMG (*Object Management Group*) [10].

La figure 4.1 présente les principaux types de diagrammes de structure proposés par UML, alors que la figure 4.2 présente les principaux types de diagrammes de comportement.

UML étant un langage de modélisation largement connu et bien documenté dans

le domaine du génie logiciel, nous nous concentrons dans le présent chapitre plus particulièrement à définir les notions nécessaires à la compréhension des mécanismes et des éléments composant les profils UML.

## 4.2 La notion de profil en UML

Un profil UML est un mécanisme d'extension, qui permet d'étendre les éléments d'UML afin d'adapter les modèles et leur contenu à un domaine particulier (par ex., domaines d'activité spécifique) ou à une plateforme particulière (par ex., .NET, J2EE).

Les extensions qu'un profil définit permettent d'ajouter des caractéristiques aux éléments standards d'UML. Ces extensions ne peuvent pas *retirer* des caractéristiques, et ce pour ne pas aller à l'encontre de la sémantique standard d'UML. Un profil se compose de stéréotypes, de valeurs étiquetées (*tagged values*) et de contraintes qui s'appliquent aux éléments de modèles UML tels que les classes, les attributs, les opérations et les activités.

### Exemple

Afin de mieux comprendre les éléments qui composent un profil, nous allons mettre en place un exemple au fur et à mesure de ce chapitre, en y intégrant divers éléments un à la fois.

Prenons un exemple d'un modèle devant représenter un tableau de bord d'une machine composée de trois boutons : `Marche`, `Action`, `Arrêt`. Chaque bouton a un état «`actif`» ou «`inactif`», et chacun a une action qui lui est propre sur la machine contrôlée par ces boutons : `Marche` = démarre la machine; `Action` = effectue l'action de la machine; `Arrêt` = termine l'exécution de la machine. La

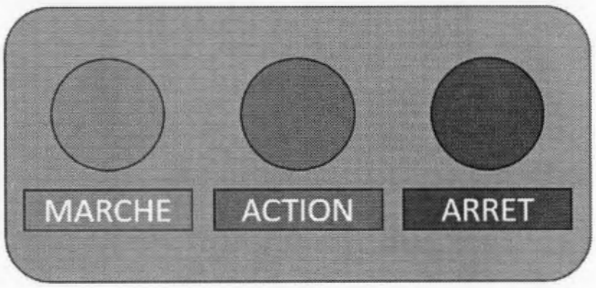


Figure 4.3: Le tableau de bord de la machine pour notre exemple de profil UML.

figure 4.3 illustre un tableau de bord possible pour une telle machine.

## 4.3 Les stéréotypes

Les stéréotypes permettent de spécifier des extensions aux métaclasses UML. Ceci permet d'ajouter des termes spécifiques de vocabulaire à un modèle. Un stéréotype peut être défini pour divers types d'éléments d'un modèle UML, par exemple, classes, associations, attributs, etc.

Exemple

Dans la figure 4.4, on remarque la présence d'une flèche noire à tête pleine entre les éléments "`«stereotype» Bouton`" et "`«metaclass» Class`". C'est la notation (graphique) UML indiquant que le stéréotype `Bouton` **étend** (*extension*) la métaclasse UML `Class`. En d'autres mots, le stéréotype `Bouton` pourra être utilisé pour *annoter des classes*. Signalons aussi qu'une extension, comme on le verra dans le prochain chapitre, peut aussi être utilisée en lien avec la métaclasse `Association`, ce qui permet alors de stéréotyper une association.

Dans notre exemple, nous avons plusieurs éléments : un tableau de bord, des boutons, des actions que les boutons effectuent. Un stéréotype, comme celui présenté

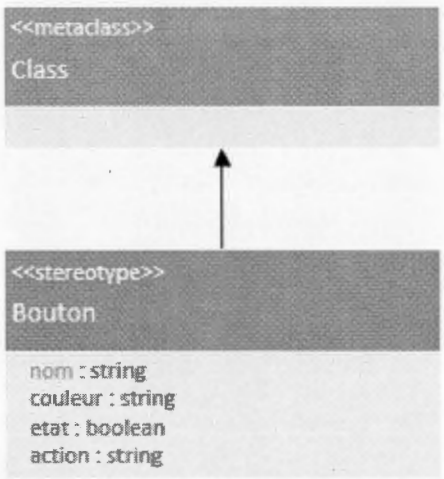


Figure 4.4: Un stéréotype «`Bouton`» permettant de spécifier les points communs aux différents boutons.

à la Figure 4.4, pourrait être défini afin de spécifier les caractéristiques communes des boutons.

Une fois défini, ce stéréotype pourra alors être utilisé pour définir différentes instances de `Bouton`, tel qu'illustré à la Figure 4.5.

## 4.4 Les valeurs étiquetées

Les valeurs étiquetées (*tagged values*) permettent d'ajouter à un élément des propriétés qui lui sont spécifiques. Plus précisément, ce mécanisme permet d'ajouter de l'information spécifique aux éléments à l'aide de paires clé/valeur. Ces couples clé/valeur sont indiqués comme illustré dans la Figure 4.5.

Exemple

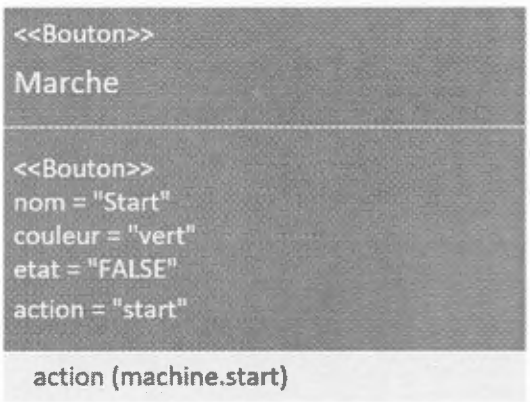


Figure 4.5: Une classe pour `Marche`, spécifiée à l'aide du stéréotype `Bouton` et de valeurs étiquetées.

## 4.5 Les contraintes

Les contraintes sont des propriétés spécifiques, qui permettent de spécifier des conditions additionnelles sur un modèle. Ces conditions peuvent être appliquées à une ou plusieurs classes, à un ou plusieurs attributs, à une ou plusieurs relations entre classes. Une contrainte est représentée, dans un modèle, sous la forme d'une note indiquant une phrase qui exprimant la contrainte entre crochets ou encore, de façon plus formelle, sous la forme d'une contrainte OCL [20].

Exemple

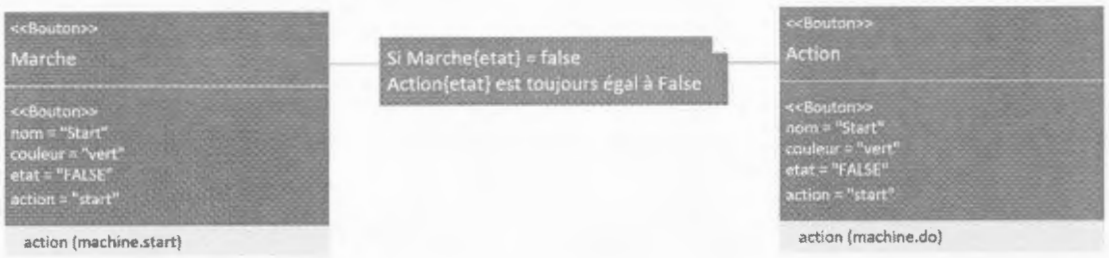


Figure 4.6: Des contraintes spécifiant des conditions sur des éléments d'un modèle.

Nous trouvons l'exemple d'une contrainte dans la Figure 4.6, entre les boutons `Marche` et `Action`. Cette contrainte énonce que si l'état du bouton `Marche` est `false`, alors l'état du bouton `Action` sera lui aussi `false`, ce qui signifie que l'action de la machine ne peut pas être effectuée si la machine est à l'arrêt. On pourra alors spécifier, dans un diagramme UML, la contrainte présentée dans la figure 4.6.

## 4.6 Les icônes

Lorsque la taille d'un modèle devient importante, il peut devenir difficile de s'y retrouver parmi les nombreux stéréotypes. Dans ce cas, l'association d'icônes aux stéréotypes peut être une solution intéressante à mettre en place dans le cas d'un profil dans un domaine d'activité spécifique. Une représentation graphique simple — une icône — peut ainsi être attribuée à un type d'élément spécifique du modèle.

Exemple

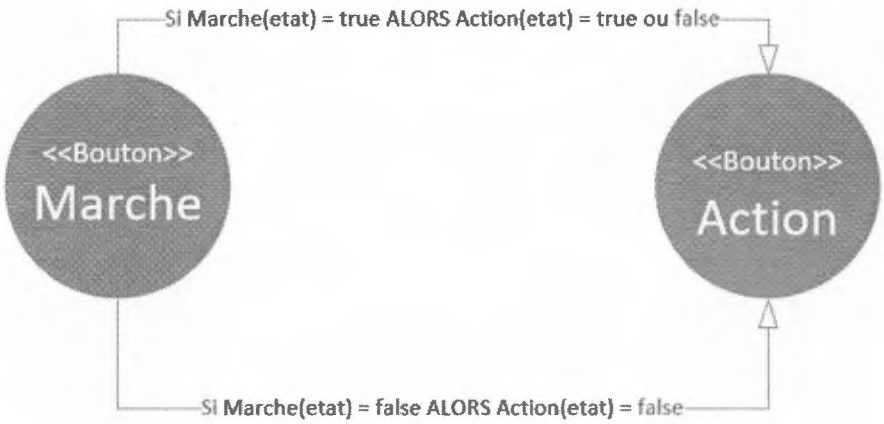


Figure 4.7: Une icône spécifique associée aux boutons.

Dans notre exemple, une machine est composée de trois boutons. On peut alors choisir, par exemple, d'attribuer une icône spécifique aux boutons afin de les différencier des autres éléments, tel qu'illustré dans la figure 4.7.

## 4.7 L'utilisation d'un profil UML pour la conception de jeux vidéos

UML est un langage de modélisation permettant de représenter divers types de systèmes et permettant de les documenter de manière rigoureuse en apportant les caractéristiques suivantes :

- UML est un langage formel et normalisé ;
- UML est manipulable avec de nombreux outils déjà approuvés ;
- UML est un support de communication performant ;
- UML est un langage qui permet le contrôle des versions et la conservation de l'information de manière efficace.

Cependant, UML est un langage de modélisation tellement étendu qu'il permet de «tout faire» — ou presque —, ce qui rend le langage difficile à appréhender. De plus, UML est prévu à l'origine pour représenter des systèmes logiciels quelconques, donc avec un vocabulaire générique.

Dans les chapitres précédents, nous avons établi une liste de points importants concernant la conception de jeux vidéos. Nous pensons qu'un profil UML serait un outil intéressant pour répondre aux besoins de modélisation d'un GDD. La mise en place d'un profil UML aurait plusieurs avantages :

- En utilisant un profil, il est possible de simplifier la compréhension des différents éléments UML utilisés pour exprimer un concept, ce qui permet de réduire la quantité de connaissances à acquérir pour se servir d'UML dans le cadre de la rédaction d'un GDD.
- Les stéréotypes permettent d'intégrer des notions spécifiques au *game design*.
- Les valeurs étiquetées permettent d'ajouter des propriétés spécifiques aux nouveaux éléments introduits par les stéréotypes.
- Des contraintes peuvent être spécifiées dans un profil afin d'éviter des erreurs.
- Des icônes peuvent être associées aux éléments du modèle pour faciliter sa compréhension.
- Les outils supportant UML sont nombreux et efficaces. Ils sont déjà présents sur le marché et répondent aux besoins de la modélisation UML.

Grâce aux ajustements et extensions permis par les profils, UML semble être un langage intéressant pour permettre la représentation graphique d'un GDD. C'est pourquoi, dans le prochain chapitre, nous proposons un profil UML permettant la description des éléments d'un GDD.

# CHAPITRE V

# UN PROFIL UML POUR LA RÉDACTION DE GDD : *GAME GENESIS*

Un *Game Design Document*, tel que décrit à la Section 2.4, permet de réunir toutes les informations de design nécessaires au développement d'un jeu vidéo. La structure du GDD est définie en respectant des bonnes pratiques et selon les besoins de design du jeu concerné.

Le *Framework Mechanics, Dynamics, Aesthetics*, décrit au Chapitre 3, permet de séparer les différents aspects du design d'un jeu vidéo. L'aspect *Mechanics* permet de représenter les éléments du jeu rattachés aux données et aux algorithmes, l'aspect *Dynamics* décrit le comportement des éléments de *Mechanics*, alors que l'aspect *Aesthetics* décrit les émotions découlant de la *Dynamics* que le jeu génère chez le joueur.

Les profils UML, décrits au Chapitre 4, permettent d'étendre les concepts présents dans UML. Mettre en place un profil UML permet de faciliter la création d'un modèle et de son contenu pour un domaine particulier. Un profil se compose de stéréotypes, de valeurs étiquetées et de contraintes, qui ensemble permettent de définir de nouveaux concepts utilisables dans la description de modèles.

## 5.1 Un aperçu de *Game Genesis*

*Game Genesis* est un profil UML que nous avons développé et qui permet d'adapter UML au domaine du design de jeux vidéos. Plus précisément, *Game Genesis* introduit de nouvelles classes et associations — par le biais de stéréotypes — afin de décrire les **éléments de *Mechanics*** d'un GDD.

Afin de rédiger un GDD, un *game designer* doit d'abord définir les bases du jeu. Durant cette étape, le *game designer* va devoir définir un certain nombre d'éléments de *Mechanics* qui correspondent aux objets présents dans le jeu ainsi que leurs actions et interactions. Ces éléments de *Mechanics* peuvent aussi bien être des objets visibles dans le jeu — comme des personnages, des ennemis, des armes, des bâtiments — mais également des éléments autour du jeu — comme des cartes, des chronomètres, des statistiques, etc. Tous ces éléments de *Mechanics* ont des caractéristiques qui leur sont propres.

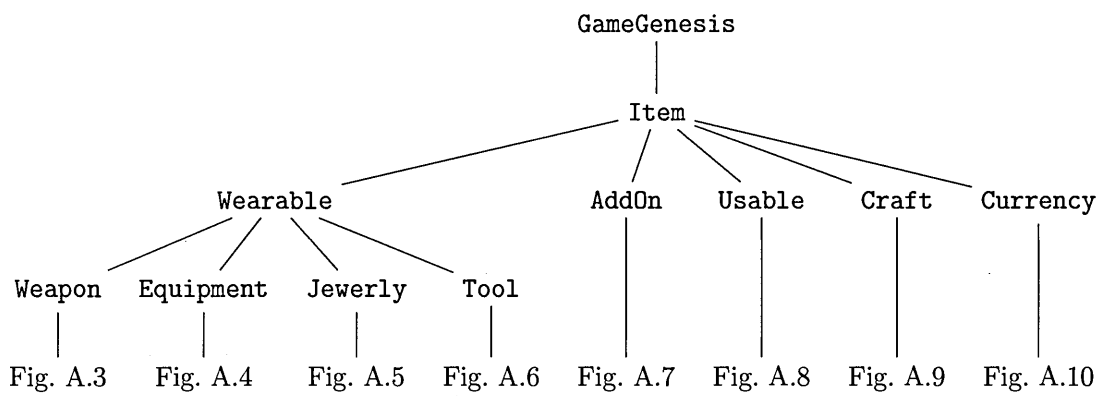


Figure 5.1: L'arbre des stéréotypes de *Game Genesis*.

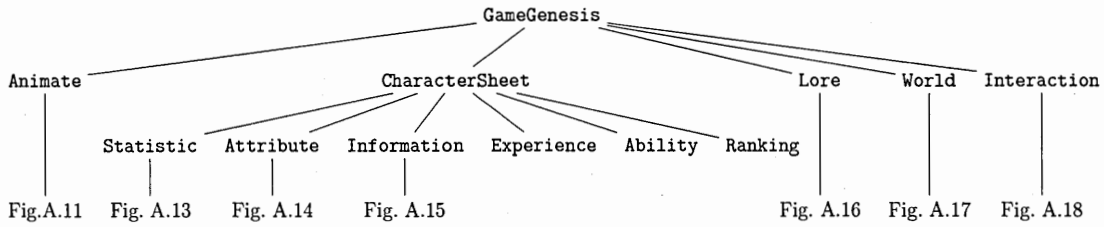


Figure 5.2: L'arbre des stéréotypes de *Game Genesis* (suite).

*Game Genesis* permet à un *game designer* de répertorier et décrire ces éléments de *Mechanics* sous forme de diagrammes de classes. Chaque catégorie de *Mechanics* est représentée par une classe, et chaque élément de *Mechanics* est aussi représenté par une classe. L'appartenance à une catégorie est définie par des relations d'héritage. Les caractéristiques des catégories et des divers éléments de *Mechanics* sont représentées par des attributs. Finalement, certaines interactions entre éléments du jeu peuvent être représentées par des associations entre classes, associations auxquelles peuvent être associées certaines contraintes.

Les figures 5.1 et 5.2 présentent une vue d'ensemble des différents stéréotypes introduits par *Game Genesis* — seuls les niveaux supérieurs de la hiérarchie des stéréotypes sont indiqués.

## 5.2 *Game Genesis* en détail

Dans les sous-sections qui suivent, nous définissons plus en détail les éléments suivants de *Game Genesis* :

— Les stéréotypes qui permettent d'étendre les classes.

— Les stéréotypes qui permettent d'étendre les associations.

— Les contraintes qui définissent des règles d'utilisation des stéréotypes.

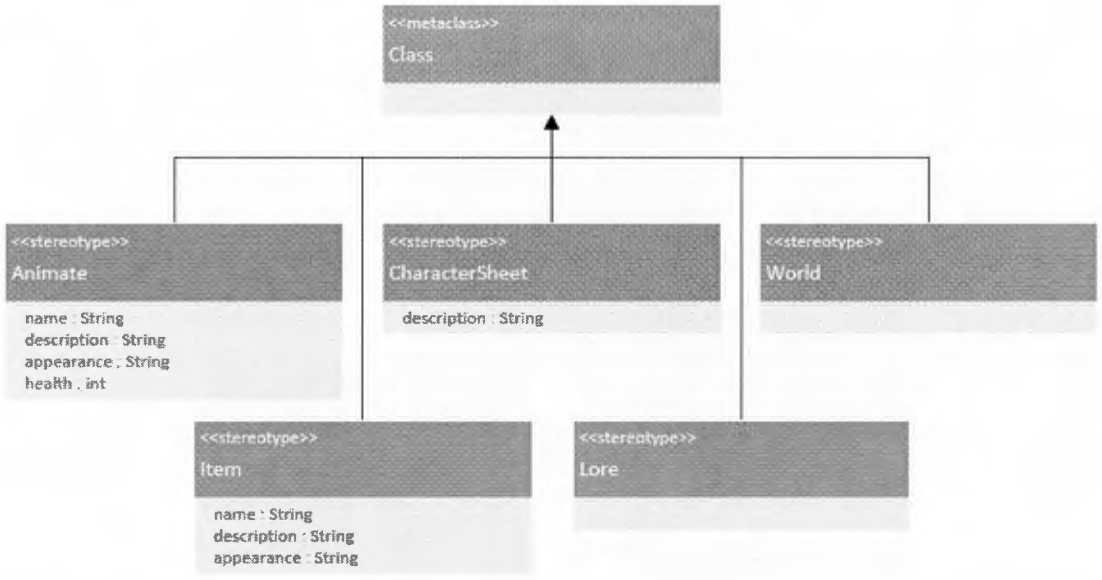


Figure 5.3: La spécification de certains stéréotypes de *Game Genesis*, obtenus par *extension* de la métaclasse `Class`.

### 5.2.1 Les stéréotypes

Dans la section 4.3, nous avons décrit le mécanisme d'extension de métaclasses afin d'introduire de nouveaux concepts par le biais de stéréotypes. Dans *Game Genesis*, nous faisons donc usage des stéréotypes afin d'étendre les classes d'un modèle de jeu et ainsi permettre la rédaction de GDD. Les racines des arbres définissant ces divers stéréotypes sont présentées aux figures 5.1 et 5.2, alors que les détails des sous-arbres sont présentés en annexe, dans une figure dont le numéro est indiqué sur ces figures.

La figure 5.3 présente les différentes « racines » de *Game Genesis*. Ce sont les premiers stéréotypes du profil qui serviront majoritairement de catégories de haut niveau pour classifier les éléments de *Mechanics* décrits dans un *Game Design Document*. Nous présentons le profil plus en détail dans l'Annexe A. Comme on

le remarque, ces stéréotypes sont des *extensions* de la métaclasse `Class`, donc ils serviront à annoter des classes, des concepts.

Dans un article de Salazar *et al.* [15], plus particulièrement dans la documentation additionnelle de cet article [16], nous avons identifié un certain nombre d'éléments de *Mechanics* typiquement présents dans un GDD. Nous avons constaté que les éléments étaient répartis dans des catégories possédant elles-mêmes certaines caractéristiques communes. Nous avons ainsi extrait les catégories suivantes :

— *Player Character*

— *Non Player Character*

— *Enemy*

— *Final Enemy*

— *Help Object*

— *Extra Object*

— *Other Object*

Ces catégories sont cependant plutôt spécifiques au jeu décrit dans cet exemple de *Game Design Document*, à savoir *Donkey Kong* [16]. Donc, ces catégories ne permettent pas forcément de représenter les éléments de *Mechanics* de jeux différents de *Donkey Kong*. Par la suite, nous avons essayé de définir des catégories et des éléments en fonction de différents *Game Design Documents* et de notre expérience personnelle de l'utilisation des jeux vidéos. Nous avons alors défini les stéréotypes présents dans les figures 5.1 et 5.2, figures qui renvoient vers une modélisation plus en détail présentée dans l'Annexe A.

Les classes UML fonctionnant avec un système d'héritage, un élément de *Mechanics* hérite donc des attributs énoncés dans les catégories parentes.

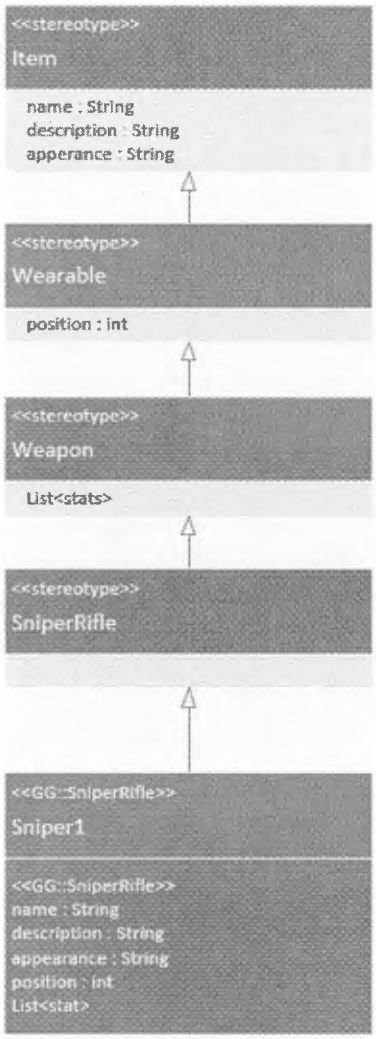


Figure 5.4: Une classe `Sniper1` utilisant le stéréotype `SniperRifle`, et donc qui hérite des attributs (directs et indirects) associés à son stéréotype `SniperRifle`.

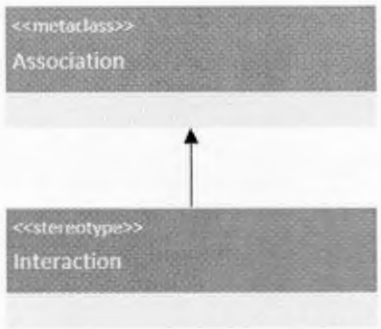


Figure 5.5: La spécification du stéréotype `Interaction` de *Game Genesis*, obtenu par *extension* de la métaclasse `Association`.

La figure 5.4 présente un exemple d'une classe `Sniper1`, un élément de *Mechanics* d'un GDD pour un jeu qui serait défini avec le profil *Game Genesis*. La classe `Sniper1` est stéréotypée avec `SniperRifle`, donc elle hérite des attributs des divers stéréotypes de la hiérarchie d'héritage de classes. Une instance de la classe `Sniper1` possèdera donc les attributs des stéréotypes `Item`, `Wearable`, `Weapon` et `SniperRifle`.

### 5.2.2 Les interactions

Les éléments de *Mechanics* ne sont pas tous des éléments isolés et une expérience de jeu ne serait rien sans les interactions que le joueur peut avoir avec les divers éléments du jeu.

La Figure 5.5 représente la spécification des stéréotypes présents dans la catégorie `Interaction`. Contrairement aux autres stéréotypes du profil *Game Genesis* qui étendent les classes, celui d'`Interaction` étend plutôt la métaclasse `Association`.

Des interactions fréquemment rencontrées sont présentées dans le gabarit de GDD de Salazar *et al.* [16], qui les présente sous forme de règles d'interaction. Ces règles sont composées de divers éléments, dont les suivants :

Tableau 5.1: Les interactions de Salazar *et al.* [16] intégrées dans *Game Genesis.*

| | |
|---|---|
| `Use` | `Drop` |
| `Move` | `Destroy` |
| `Trade` | `Stack` |
| `Attach` | `Attack` |
| `Wear` | |

— Deux éléments (ou plus) de *Mechanics* ;

— Une interaction entre ces éléments ;

— Une contrainte appliquée à cette interaction (optionnelle).

Les interactions de Salazar *et al.* que nous avons intégrées à *Game Genesis* sont présentées dans le tableau 5.1 — voir aussi la Figure A.18.

### 5.2.3 Quelques contraintes

Dans un profil UML, il est fréquent d'imposer des contraintes d'utilisation sur les éléments du profil afin de le rendre plus précis et d'éviter des problèmes de compréhension ou d'incohérence du modèle. Dans *Game Genesis*, il est difficile d'établir un grand nombre de contraintes, le profil devant s'appliquer à de nombreux types de mécaniques de jeu.

Voici quand même une brève liste de contraintes envisageables dans un profil tel que *Game Genesis* :

— Une interaction entre `Player` et `Weapon` ne peut être que `Grab` ou `Equip` ;

— Une interaction entre `Player` et `Equipment` ne peut être que `Grab` ou `Equip` ;

— Une interaction entre `AddOn` et `Weapon` ou `Equipment` ne peut être que

`Attach`;

Certaines contraintes peuvent être situationnelles comme celles ci-dessous :

— Si le jeu est `PvP` (*Player versus Player*) alors `Player «attack» Player` est possible,

— Si le jeu n'est pas `PvP` alors `Player «attack» Player` n'est pas possible.

## 5.3 Conclusion

Dans ce chapitre, nous avons tenté d'exprimer les concepts de base d'un profil UML pour la rédaction d'un *Game Design Document*. Dans son état actuel, ce profil, *Game Genesis*, est cependant plus une «preuve de concept» qu'une version complète qui pourrait couvrir tous les besoin d'un *game designer*. *Game Genesis* est incomplet car :

— seuls les aspects *Mechanics* sont traités ;

— les catégories sont non exhaustives ;

— le profil est spécifié de façon non formelle, aucun outil UML «officiel» n'ayant été utilisé pour le spécifier.

Il serait quand même intéressant de voir comment *Game Genesis* peut être utilisé dans le cadre de la rédaction d'un *Game Design Document* pour un jeu existant. C'est ce que nous faisons dans le prochain chapitre.

# CHAPITRE VI

# UN EXEMPLE D'APPLICATION DU PROFIL *GAME GENESIS* : PUBG

PUBG (*PlayerUnknown's Battlegrounds*) est un jeu vidéo, sorti en 2017, qui compte des millions de joueurs à travers le globe. Développé par *PUBG Corporation* (filiale de B*luehole, Inc.*, plus récemment de *Krafton Game Union*), PUBG est un des premiers jeux de type «*Last man standing*» et *standalone*,[1] avec *Fortnite* (*Epic Games*). Ces deux jeux ont été les premiers *standalone* à populariser le *Battle Royale* et à le mettre à la portée de tous sans devoir développer ou installer du contenu additionnel dans des jeux existants.

## 6.1 L'origine des jeux *Battle Royale*

Les jeux de type *Battle Royale* trouvent leurs racines dans le roman «*Battle Royale*» [18] et son adaptation cinématographique.[2] En gros, l'histoire va comme suit. Un programme militaire de simulation de combat a lieu dans une république socialiste d'Extrême-Orient coupée du monde extérieur, extrêmement stable politiquement, où les habitants n'ont aucun droit civique. Le programme militaire

1. Jeu *standalone* : jeu indépendant, c'est-à-dire qui n'est pas un contenu additionnel ou un module d'un autre jeu.

2. `https://www.imdb.com/title/tt0266308`

débute par un tirage au sort annuel parmi 50 classes d'élèves de troisième année. La classe tirée au sort est déplacée vers une zone de combat. Au début de l'expérience, tous les élèves sont réunis afin de recevoir un briefing rapide, puis se voient attribuer un sac contenant un seul objet (par ex., arme à feu, arme blanche, fourchette, corde de luth, etc). Les élèves sont ensuite transportés dans une zone donnée et livrés à eux-mêmes, en ayant pour seule consigne qu'aucune règle n'est imposée... et qu'un seul survivant par groupe pourra être gagnant de l'expérience ; les autres devront être exterminés, et ce par n'importe quel moyen. Le champion obtient alors le droit de vivre aux frais de l'État pour le reste de ses jours et la reconnaissance comme « Héros du pays ».

## 6.2 L'essor de la popularité du *Battle Royale*

Bien que connu depuis l'année 2000, l'intérêt pour le principe du *Battle Royale* n'explose que plus tard, avec le succès de la saga *Hunger Games*[3] qui met en place l'histoire de Katniss Everdeen, une jeune adulte qui participe de son plein gré aux *Hunger Games*. Dans un état totalitaire séparé en castes regroupées dans des districts, deux enfants ou adolescents sont choisis au hasard dans chaque district afin de devenir les Tribus. Ils sont alors réunis dans la capitale et lâchés dans une arène afin de participer à une télé-réalité de match à mort diffusée partout dans le pays.

Le succès des *Hunger Games* a amené les développeurs du jeu *Arma II* à créer un mode de jeu basé sur les mêmes règles. Ce mode sera alors appelé *DayZ* et deviendra par la suite un jeu *standalone*. Un mode voit également le jour : *Battle Royale*. Développé par Brendan Greene tout d'abord pour *Arma II* puis pour *Arma III*, ce mode gagne suffisament en popularité pour qu'il donne naissance au

3. https://www.imdb.com/title/tt1392170/

projet de jeu *PlayerUnknown's Battlegrounds* (PUBG) — *PlayerUnknown* étant le pseudonyme en ligne utilisé par Brendan Greene.

## 6.3 Les règles de PUBG

Les règles du jeu PUBG vont comme suit. Un avion parcourt une ligne droite au-dessus d'un territoire, et les 100 joueurs de la partie doivent sauter de cet avion afin d'être parachutés à l'endroit de leur choix — à condition qu'il soit à portée de parachutage. Une fois arrivés au sol, les joueurs partent à la recherche d'armes et d'équipements dans des bâtiments et doivent tuer les autres joueurs jusqu'à ce qu'il ne reste qu'un seul survivant, lequel gagne alors la partie. Il est également possible de jouer en équipes de deux ou quatre joueurs, où la dernière équipe encore vivante est l'équipe gagnante. Afin de limiter la durée d'une partie, une zone circulaire est définie sur la carte une fois que tous les joueurs ont atterri, et cette zone se réduit au cours de la partie — ce qu'on appelle la *Blue Zone*. Les joueurs en dehors de la zone subissent une certaine quantité de dégâts s'ils restent à l'extérieur de cette zone, et ces dégâts augmentent au fur et à mesure que le cercle se resserre. Une autre zone apparaît également sur la carte : la *Red Zone*. C'est une zone de bombardement qui change d'emplacement régulièrement dans une partie. Lorsque cette zone est active, plusieurs bombes tombent au sol, et une seule bombe suffit à tuer un joueur.

## 6.4 Le *Battle Royale* dans le monde du jeu vidéo

Ces dernières années, une grande quantité de jeux vidéos de type *Battle Royale* ont vu le jour. De *Fortnite* (*Epic Games*) à PUBG, en passant par *Apex Legends* (*Respawn Entertainement*), *Ring of Elysium* (*Tencent Games*) pour ne citer que les plus populaires. Des dizaines de jeux voient le jour ou se réinventent afin de

coller à la mode du *Battle Royale*. Les plus grandes licences de FPS (*First-Person Shooter*) s'alignent également à cette tendance, et c'est ainsi que voient le jours les modes de jeu *Z1BR* (*H1Z1*), *Firestorm* (*Battlefield V*), *Blackout* (*Call of Duty Black Ops IV*). Mais cette offre répond à une demande phénoménale du public pour ce mode de jeu qui s'impose dans le marché à la même place que les jeux de type MOBA (*Multiplayer Online Battle Arena*), qui étaient des grands favoris du public depuis une dizaine d'années, notamment avec les jeux *League Of Legends* ou *Dota 2*. Avec le temps, les jeux de *Battle Royale* tentent de se réinventer et de trouver de nouveaux publics en gardant le principe du *Battle Royale*, mais en explorant d'autres univers. C'est ainsi que naissent *Last Tide* et son univers aquatique immergé, *Fall Guys : Ultimate Knockout* et son univers coloré où de petits personnages à la physique étrange se battent pour une couronne, *Cuisine Royale*, qui était à l'origine une blague de développeurs mais qui a rencontré un immense succès, dans lequel les personnages se battent à l'aide d'ustensiles de cuisine.

## 6.5 Une description partielle des éléments de *Mechanics* du jeu PUBG

### 6.5.1 Les informations décrivant le jeu PUBG

Les informations concernant les objets de PUBG sont difficile à trouver. Cependant, certains sites internet se sont spécialisés dans l'extraction (*mining*) d'informations à partir des fichiers du jeu. C'est le cas du *Gamepedia* spécialisé pour PUBG (`https://pubg.gamepedia.com/`), un Wiki régulièrement mis à jour et possédant assez d'informations pour permettre d'identifier les objets du jeu et leurs caractéristiques. Les informations disponibles sont toutefois souvent sujettes à modifications au fur et à mesure des mises à jour du jeu. Cependant, l'exactitude des informations n'a pas une importance cruciale dans le cadre du présent travail.

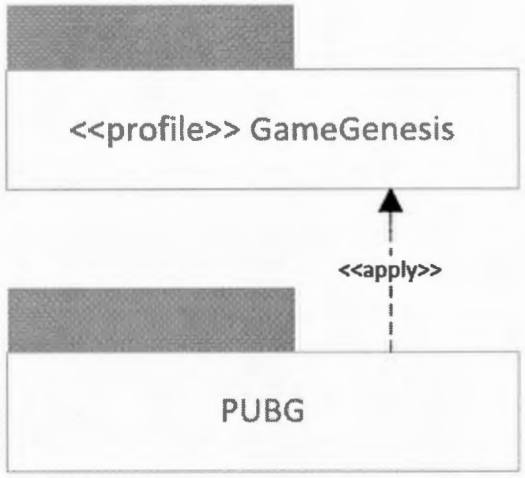


Figure 6.1: Un modèle pour le jeu PUBG qui va utiliser le profil *GameGenesis*.

### 6.5.2 Une application de *Game Genesis* à la modélisation de PUBG

Afin d'utiliser le profil *Game Genesis* dans un modèle, il faut indiquer l'application du profil dans le modèle en question, et ce avec l'association stéréotypée UML «`apply`», tel que représenté dans la Figure 6.1.

Les stéréotypes présents dans *Game Genesis* permettent de classifier les éléments de *Mechanics* pour la rédaction d'un GDD. Afin de mieux comprendre le fonctionnement de *Game Genesis*, nous avons extrait une partie du profil et la présentons dans la Figure 6.2. Cette partie du profil inclut les stéréotypes qui sont utilisés dans les exemples présentés dans la suite du chapitre. De nombreux attributs ont été ajoutés dans les stéréotypes. Ceux-ci correspondent aux attributs qui auraient pu être ajoutés par un *game designer* dans le profil. Ajouter de tels attributs permet d'inclure automatiquement les attributs dans les classes créées à partir du profil UML par l'héritage.

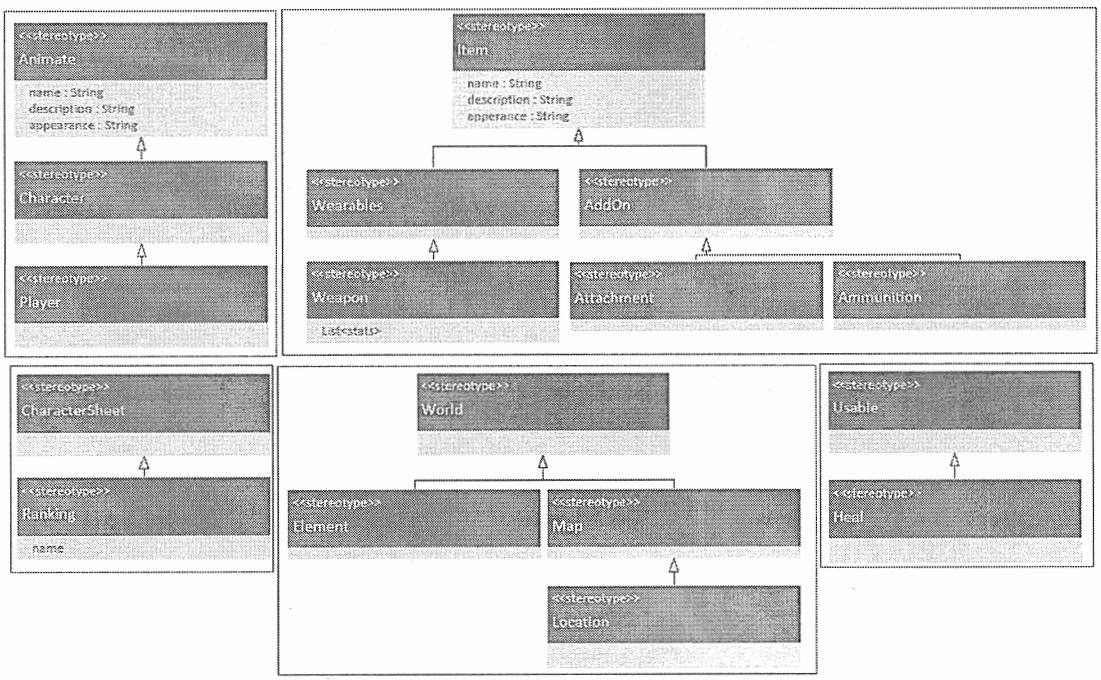


Figure 6.2: Les principaux stéréotypes de *Game Genesis* utilisés dans notre exemple, le modèle pour PUBG.

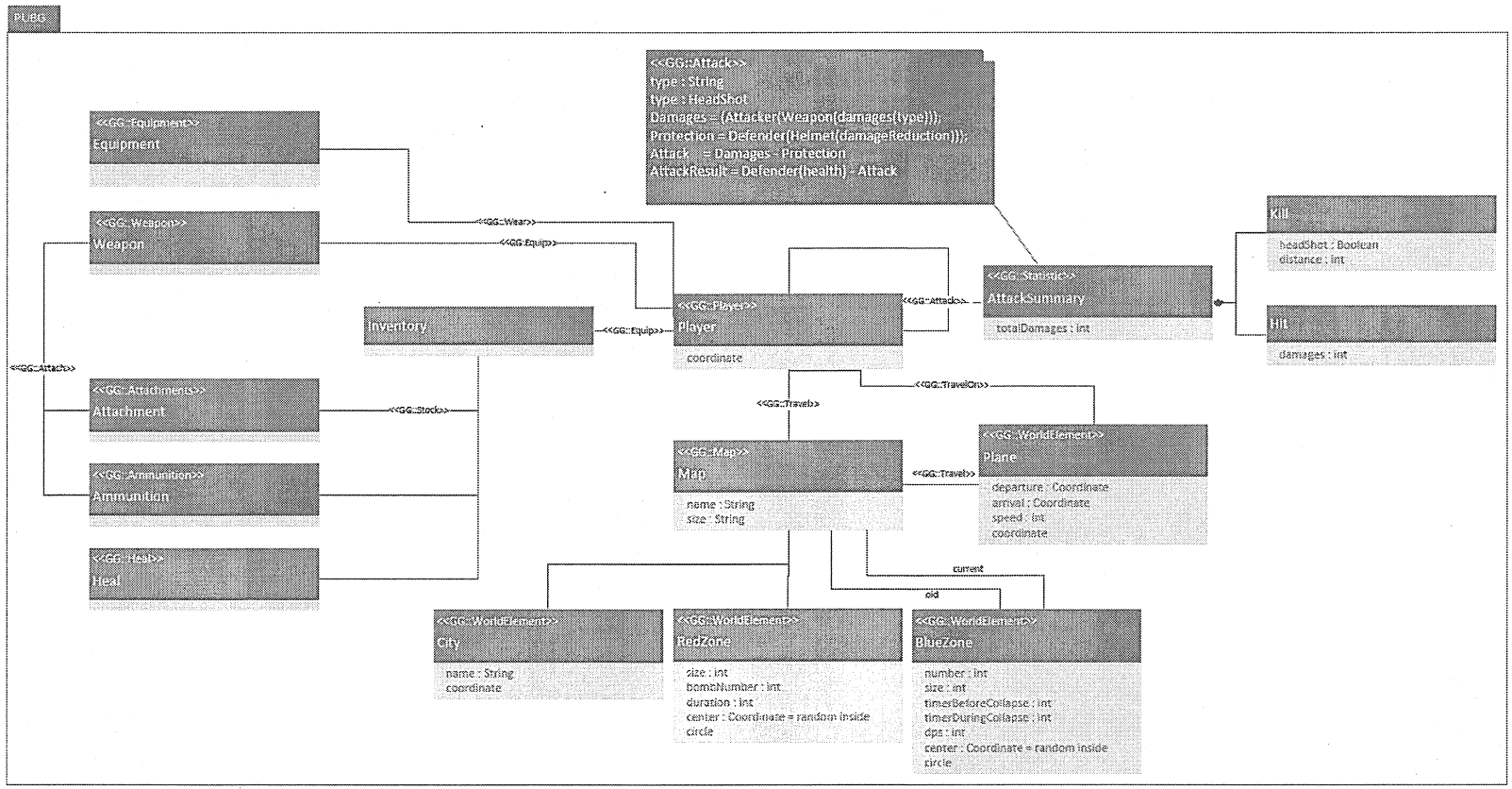


Figure 6.3: Un modèle UML (partiel) utilisant le profil *GameGenesis* pour décrire les éléments de *Mechanics* de PUBG.

### 6.5.3 La modélisation des éléments de *Mechanics* de PUBG

La figure 6.3 présente un modèle décrivant, de façon partielle, les éléments de *Mechanics* pour une partie de PUBG. Décrire l'intégralité du système serait extrêmement long et complexe. Nous nous sommes donc concentré sur certains des éléments plus intéressants.

Lors de la description du jeu présenté à la section 6.3, nous avons cité quelques éléments de mécaniques de jeu présents dans PUBG :

- Un avion parcourt un territoire et contient initialement 100 joueurs ;
- Les joueurs ramassent des armes ;
- Un joueur doit tuer les autres joueurs ;
- Une *BlueZone* délimite une zone à l'extérieur de laquelle un joueur subit des dégâts ;
- Une *RedZone* délimite une zone de bombardement.

Ces éléments sont représentés dans un modèle UML utilisant *Game Genesis* présenté à la Figure 6.3 :

- un objet `Plane` (avion) qui `Travel` (voyage) à travers une `Map` (carte) ;
- une `Weapon` (arme) qui `Equip` un `Player` ;
- un `Player` qui `Attack` un `Player` ;
- une `BlueZone` qui a une `size` (taille) et qui inflige des `damages` (dommages) ;
- une `RedZone` qui possède un `BombNumber` (nombre de bombes) et une `duration` (durée de vie).

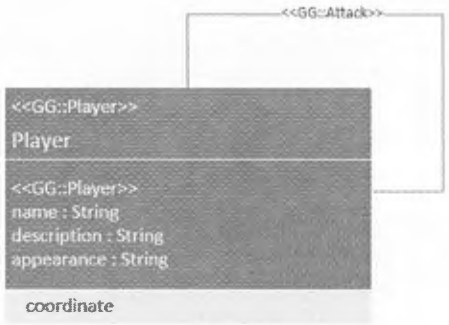


Figure 6.4: Le modèle pour un `Player` qui en attaque un autre.

### 6.5.4 La modélisation d'une attaque d'un joueur par un autre

Dans une partie de PUBG, le but ultime pour un joueur est d'éliminer les autres joueurs et de sortir vainqueur en tant que dernier survivant. La notion d'attaque entre joueurs est donc essentielle dans la description des éléments de *Mechanics* de PUBG. Ici, elle est représentée par une association récursive sur `Player` avec l'interaction `Attack` — présentée dans la Figure 6.4.

Afin de définir plus précisément l'interaction `Attack`, le *game designer* peut décider d'y intégrer des attributs et des méthodes afin de mieux représenter et mieux documenter une attaque entre deux joueurs. Par définition, une attaque sur PUBG consiste en le fait qu'un joueur tire sur un autre joueur pour lui faire subir des dégâts. Afin de calculer ces dégâts, un certain nombre d'informations sont nécessaires, comme la quantité de dégâts ou les protections possible contre les dégâts. Dans un FPS, il également est possible d'avoir plusieurs types de dégâts en fonction de l'emplacement de l'attaque. C'est le cas dans PUBG, qui différencie les dégâts en trois types classés du plus fort au moins fort : tête > torse > autres.


Figure 6.5: Le calcul des dégâts d'une attaque.

En outre, le calcul des dommages s'effectue en prenant en compte les protections portées par le joueur adverse. Afin d'exprimer le calcul des dégâts nous avons mis en place la formule suivante :

$$Damages = Attacker(Weapon(damages(type(value))))$$
$$Protection = Defender(Protection(damageReduction))$$
$$Calcul = Defender(health) - (Damages - Protection)$$

Cette formule de calcul des dégâts est présente dans l'association entre les deux joueurs sur laquelle un stéréotype `Attack` est appliqué (Figure 6.5).

On considère alors que le *game designer* a raffiné la classe utilisant le stéréotype `Attack` de *Game Genesis* afin que toutes les actions d'attaque soient régies par cette formule.

Appliquer ce calcul des dégâts pourrait par exemple donner la simulation suivante :

> Un `Player`, appelé `Player25`, possédant une arme de type `Kar98k`, effectue un tir à la tête sur un autre joueur, appelé `Player26`, portant un `HelmetT2`.

$$Damages = Attacker : \texttt{Player25}(\texttt{Weapon} : \texttt{Kar98k}(damages(type : HeadShot(197.50))))$$

$$Protection = Defender : \texttt{Player26}(Protection : \texttt{HelmetT2}(damageReduction : 40\%))$$

$$Calcul = Defender : \texttt{Player26}(health : 100) - (197.50 - (Protection : 40\%))$$

$$Result = -18.5$$

Donc, dans cet exemple, l'attaque génère 118.5 points de dégâts, laissant `Player26` avec un total de vie de -18.5. Ce total négatif indique alors la mort de `Player26`.

La mort d'un `Player` si sa vie est inférieure à 0 peut être exprimée par les contraintes suivantes :

```
Player(health) : is max 100.
IF Player(health) < 0
{
  IF Gamemode is "solo" ==> Player is dead and eliminated.
  IF Gamemode is "duo" OR "squad" ==> Player is knocked.
}
```

## 6.6 Conclusion

Dans ce chapitre, nous avons illustré l'utilisation du profil *Game Genesis* dans le cadre d'un jeu existant, PUBG. On a vu qu'il était possible de modéliser certaines parties de ce jeu à l'aide des stéréotypes. Les stéréotypes de classes sont utiles pour représenter les éléments de *Mechanics* du jeu, alors que les stéréotypes d'interaction permettent d'indiquer des relations clés entre éléments du modèle, relations pour lesquelles des informations doivent être conservées.

La liste des stéréotypes étant encore à l'état préliminaire, *Game Genesis* n'est pas encore prêt à être considéré comme viable pour tous les types de mécaniques de jeu. Cependant, il peut être considéré comme une piste de recherche. Afin que *Game Genesis* puisse être considéré comme une solution pour la rédaction d'un *Game Design Document*, il faudra donc que celui-ci soit étendu et, surtout, mis en œuvre dans un outil UML.

# CONCLUSION

Un projet de développement informatique est une tâche de longue haleine rythmée par de nombreux changements, et la description des concepts et la conception sont essentielles au bon déroulement du projet ainsi qu'à sa réussite. Ceci encore plus marqué dans le domaine des jeux vidéos car le développement d'un jeu implique non seulement des développeurs mais également de nombreux autres corps de métier : artistes (planches pour illustrer les éléments graphiques), musiciens (bande originale du jeu), modeleurs et animateurs 3D (éléments graphiques), scénaristes (cadre et histoire du jeu), etc. Réussir à apporter les informations nécessaires à tous ces corps de métier est un défi que le *game designer* affronte au quotidien.

Un document de *game design* est un pilier d'un projet de développement d'un jeu. Sa rédaction est complexe, chronophage et nécessite une rigueur exemplaire des *game designers.* Cependant, aucun gabarit préconçu n'existe pour répondre à tous leurs besoins. Chaque projet est différent, chaque jeu possède des spécificités, chaque équipe de développement est différente et cela empêche l'établissement d'un modèle générique de *Game Design Document.*

Dans la littérature, nous avons identifié plusieurs listes de bonnes pratiques de design, chacune essayant de généraliser ses conseils pour tous les genres de jeu. C'est le cas du *framework MDA*, qui propose un découpage d'un jeu afin de structurer les différents éléments dans trois catégories et lier ces éléments pour qu'ils fonctionnent ensemble.

Définir tous les éléments d'un système et décrire leurs interactions est une approche déjà présente dans le développement logiciel classique. C'est ce que l'on

retrouve notamment à travers la modélisation UML. Cependant, UML est un langage générique, conçu pour le développement informatique et il est donc compliqué pour les autres corps de métier d'acquérir les connaissances nécessaires à sa compréhension. Il est cependant possible de créer des profils UML afin d'adapter les modèles UML et les étendre pour intégrer un vocabulaire spécifique à un domaine d'activité particulier.

Une fois le vocabulaire adapté à la conception de jeux vidéos, UML a la capacité d'apporter plusieurs avantages. UML est un langage formel, permettant la description de systèmes et de processus, efficace, extensible et largement documenté. Les outils qui l'accompagnent sont fiables et respectent une rigueur qui permet la réutilisation des modèles et en respectant le contenu des données. Sa formalisation permet également d'assurer la pérennité des modèles dans le temps et leur cohérence, peu importe le nombre de modifications et la durée de leur utilisation.

C'est ce que nous recherchions dans notre problématique : apporter une description fiable, modifiable, compréhensible et précise dans la conception d'un jeu vidéo et la rédaction de son *Game Design Document*. Et c'est dans cette optique que nous avons conçu *Game Genesis*, un profil UML intégrant un vocabulaire adapté à la conception de jeux vidéos dans des modèles UML. Pour ce faire, nous avons étudié différents *Game Design Documents* afin de définir des caractéristiques communes à ces documents. Nous nous sommes concentrés sur la modélisation des éléments de *Mechanics* présents dans ces documents afin d'établir une liste des éléments qui se retrouvent dans de nombreux genres et donc de nombreux projets de développement de jeux vidéos. Une fois cette liste établie, nous avons proposé une liste de stéréotypes correspondant à ces éléments, liste dont les éléments ont été organisés selon une structure hiérarchique afin de les classifier dans des catégories conceptuelles appropriées. Afin de représenter, cette hiérarchie nous avons utilisé le mécanisme d'héritage d'UML. Ceci nous a permis de définir des caractéristiques

dans les catégories et les éléments qui leur sont associés héritent des attributs.

Afin de décrire les interactions entre les éléments de *Mechanics*, nous avons utilisé des associations. Les stéréotypes de *Game Genesis* comprennent donc une catégorie `Interaction` dont les éléments s'appliquent sur les associations UML. Ces stéréotypes décrivent des interactions entre éléments et peuvent porter des attributs et méthodes.

Finalement, nous avons exploré l'utilisation du profil *Game Genesis* dans un exemple à travers le jeu *PlayerUnknown's BattleGrounds* (PUBG). Nous avons modélisé certains des concepts clés d'une partie de ce jeu en utilisant un diagramme de classes sur lequel était appliqué le profil *Game Genesis*. Donc, plus spécifiquement, avec ce modèle, nous avons décrit une partie des éléments de *Mechanics* du jeu PUBG et certaines de leurs interactions.

*Game Genesis* ne se concentre que sur la description des éléments de *Mechanics* d'un jeu. Or, ce n'est qu'une partie du design d'un jeu, de la rédaction d'un *Game Design Document* et du *Framework MDA*. Nous nous sommes concentrés sur ces éléments de façon à limiter l'envergure du travail, car la représentation de tous les éléments du *framework MDA* aurait été trop complexe à réaliser dans le cadre d'un travail de maîtrise. On peut toutefois envisager que d'autres types de diagrammes UML puissent être utilisés pour le *design* de jeux, notamment pour la description des éléments de *Dynamics*.

La représentation des éléments et concepts d'un jeu dans un diagramme de classes ouvre de nombreuses possibilités pour la rédaction de *Game Design Documents*. En effet, il est possible de réutiliser les modèles, et UML étant un langage informatique, il est possible de le manipuler et de générer d'autres documents. En réutilisant les modèles, il serait possible d'extraire des informations des modèles afin de générer le début d'un *Game Design Document* décrivant les éléments de

*Mechanics* du jeu en question. Il serait également possible de générer un squelette d'application à partir des modèles, un diagramme de classes étant tout indiqué pour cette transformation. Nous pourrions aussi imaginer un outil de gestion des versions permettant de faire un différentiel entre les différentes versions des modèles. Il serait alors possible de stocker les justifications d'une modification effectuée et documenter ces modifications pour le futur. Par contre, toutes ces possibilités sont pour l'instant des pistes car *Game Genesis* est aujourd'hui à l'état de preuve de concept. Ces possibles évolutions demandent que notre profil *Game Genesis* soit incorporé de façon effective dans un outil UML existant, ce que nous n'avons pas eu le temps de faire, et qui serait une piste intéressante pour un travail futur.

Finalement, nous espérons que cette recherche aura permis d'identifier des pistes intéressantes et nouvelles pour concevoir un outil adapté au domaine du développement de jeux vidéos, avec comme objectif d'apporter une représentation graphique des éléments de *Mechanics* — et possiblement de *Dynamics* — et permettant de structurer et de documenter le processus de conception et d'accompagner les *game designers* dans leur tâche.

# ANNEXE A

# DIAGRAMMES DE CLASSES DU PROFIL *GAME GENESIS*

## A.1 Racine

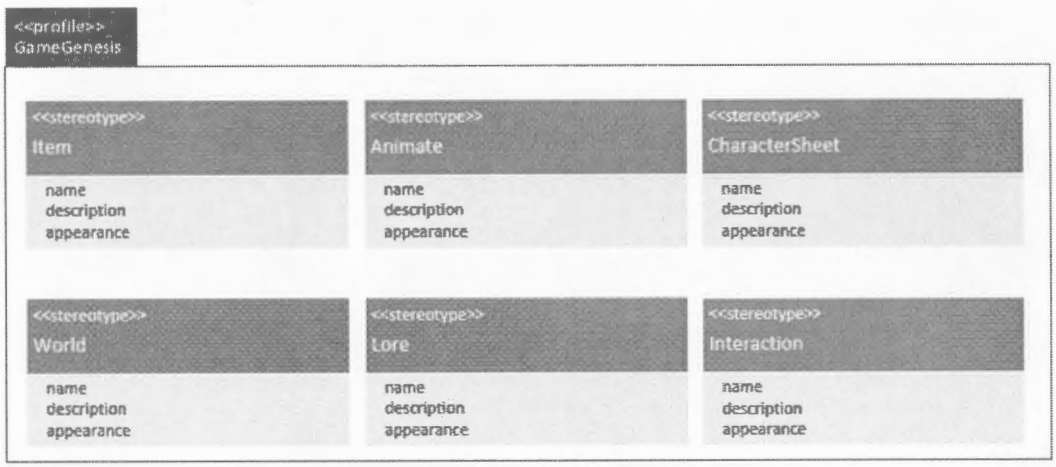


Figure A.1: Racine du modèle

## A.2 Item

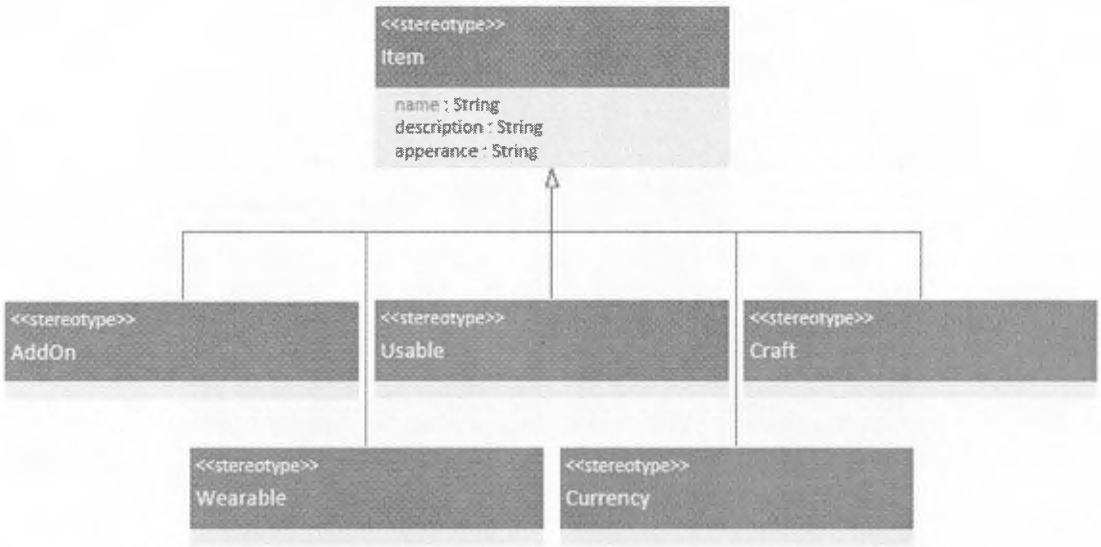


Figure A.2: Item

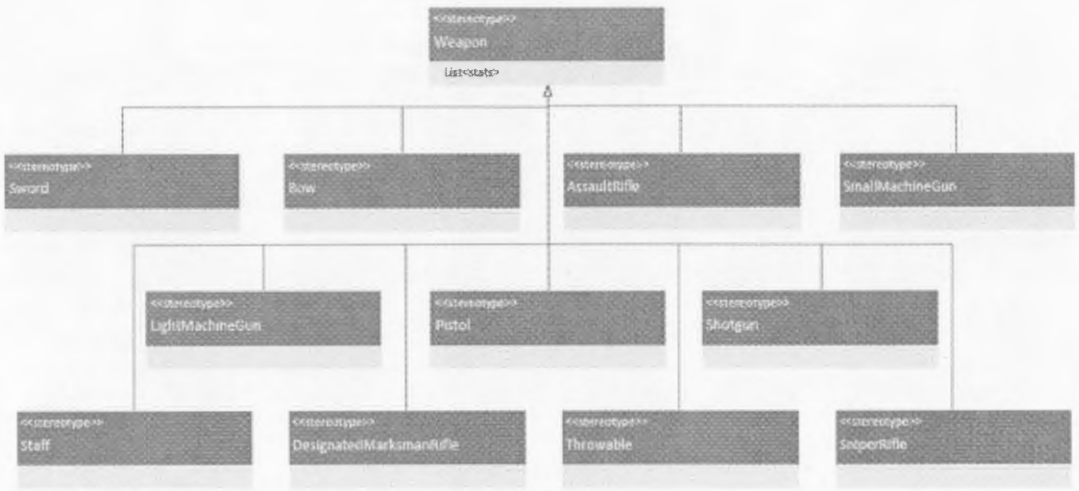


Figure A.3: Item - Wearable - Weapon

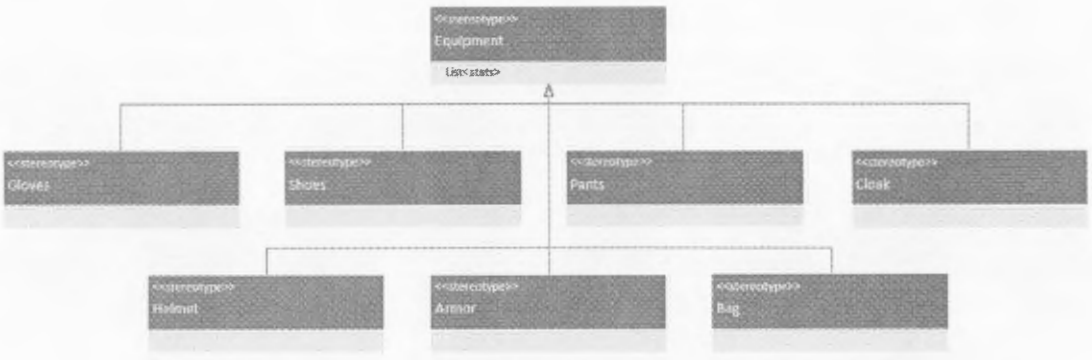


Figure A.4: Item - Wearable - Equipment

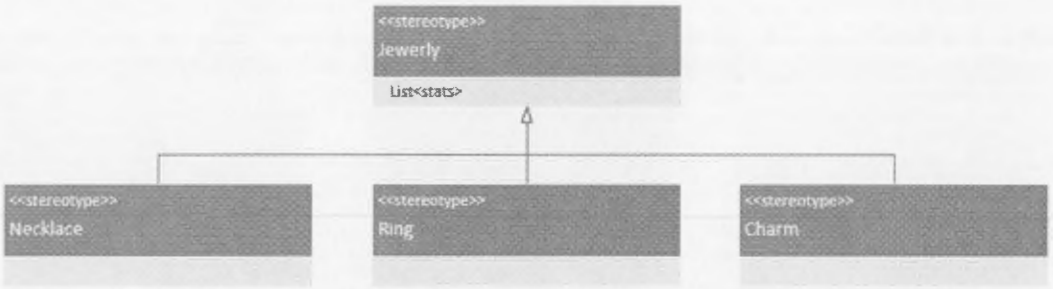


Figure A.5: Item - Wearable - Jewerly

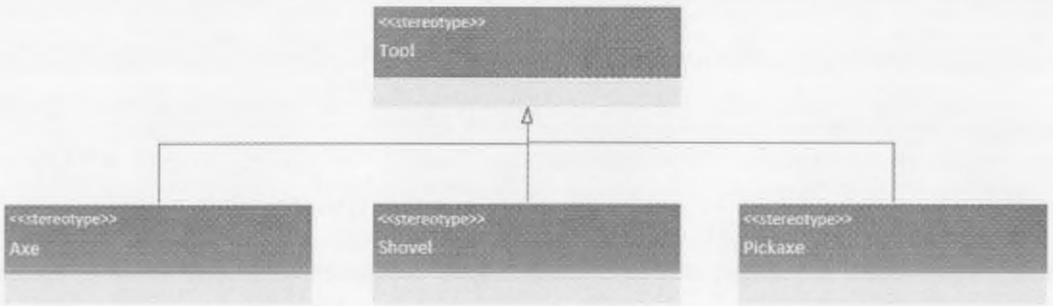


Figure A.6: Item - Wearable - Tool

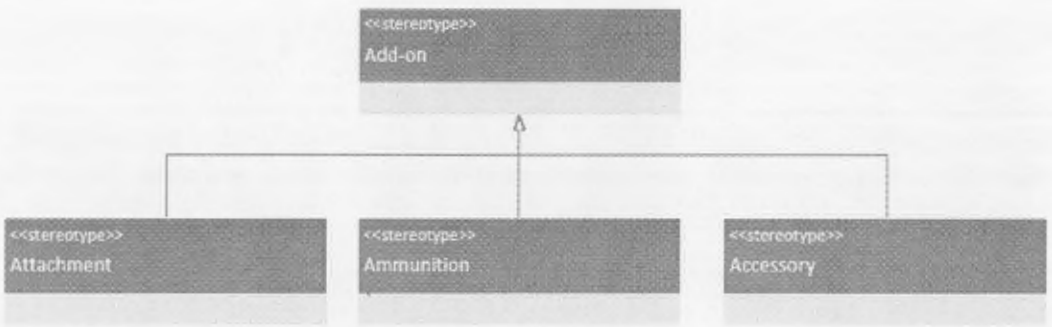


Figure A.7: Item - AddOn

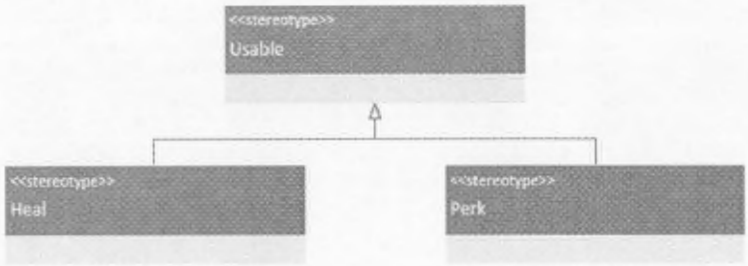


Figure A.8: Item - Usable

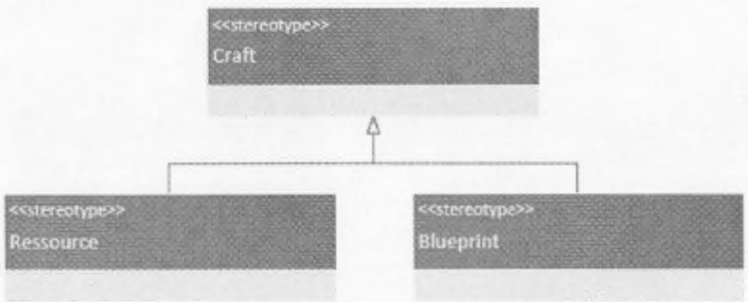


Figure A.9: Item - Craft

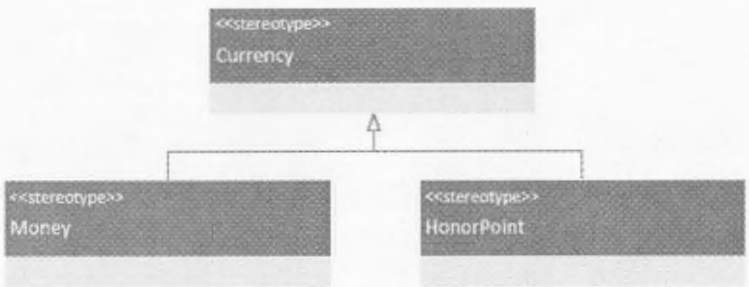


Figure A.10: Item - Currency

## A.3 Animate

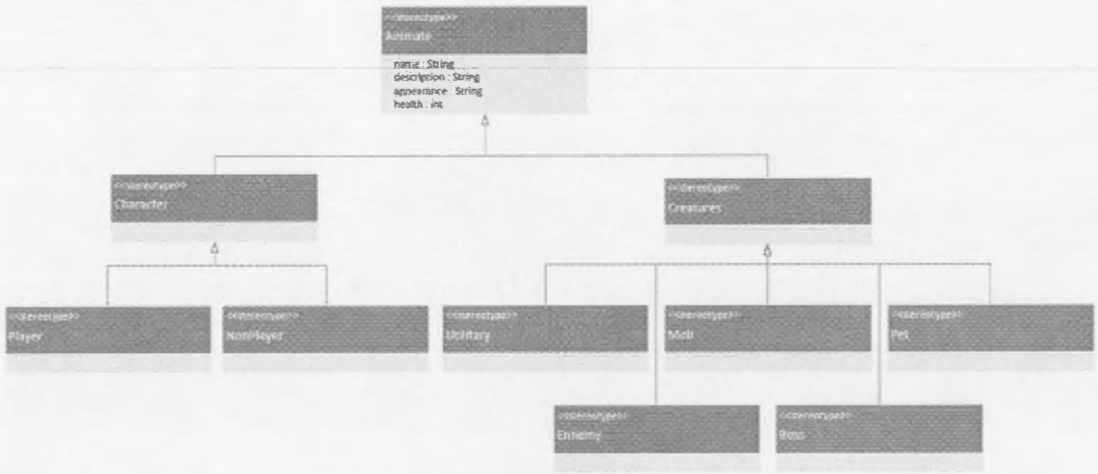


Figure A.11: Animate

## A.4 Character Sheet

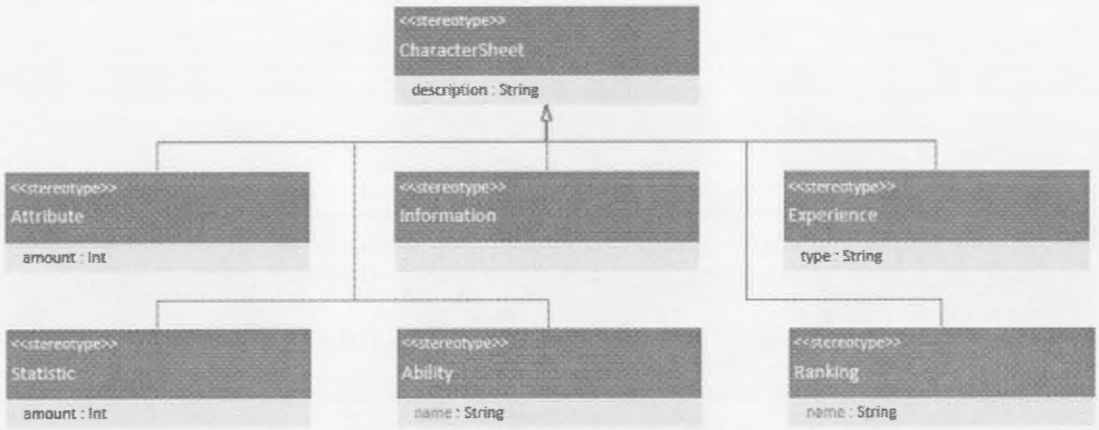


Figure A.12: CharacterSheet

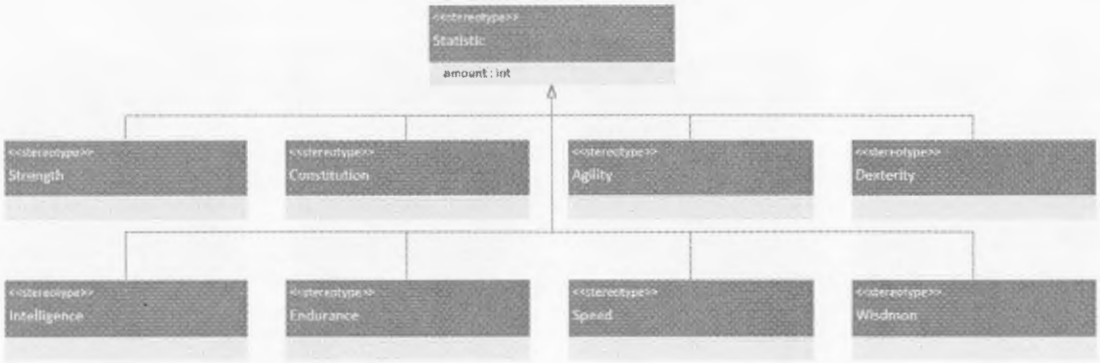


Figure A.13: CharacterSheet - Statistic

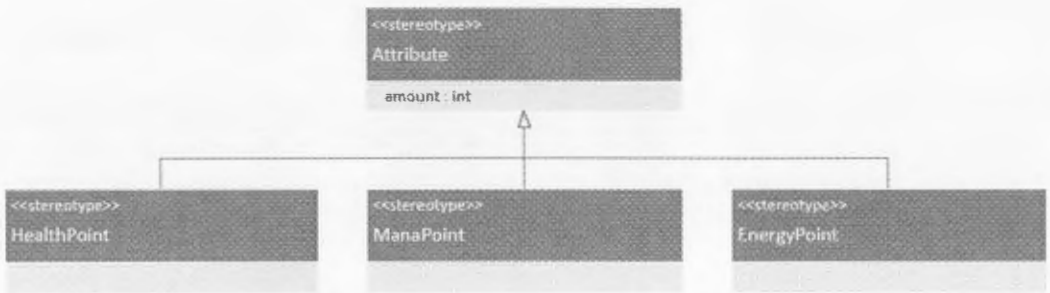


Figure A.14: CharacterSheet - Attribute

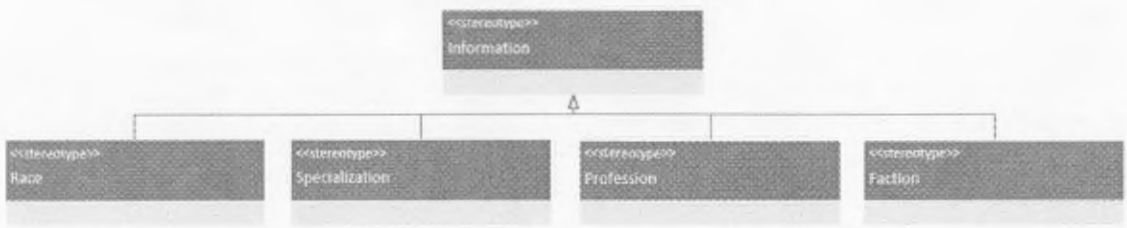


Figure A.15: CharacterSheet - Information

## A.5 Lore

Remarque : Le terme «*lore*» selon le *Cambridge Dictionary* (`https://dictionary.cambridge.org`) est défini comme suit :

> *traditional knowledge and stories about a subject :*

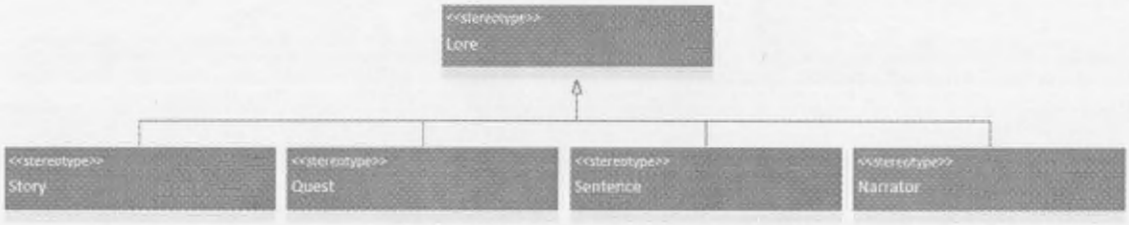


Figure A.16: Lore

## A.6 World

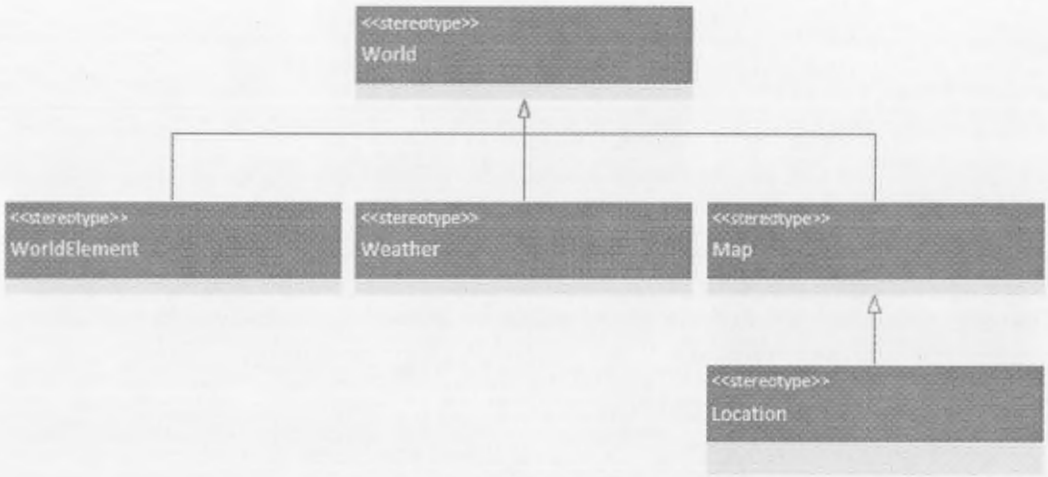


Figure A.17: World

## A.7 Interaction

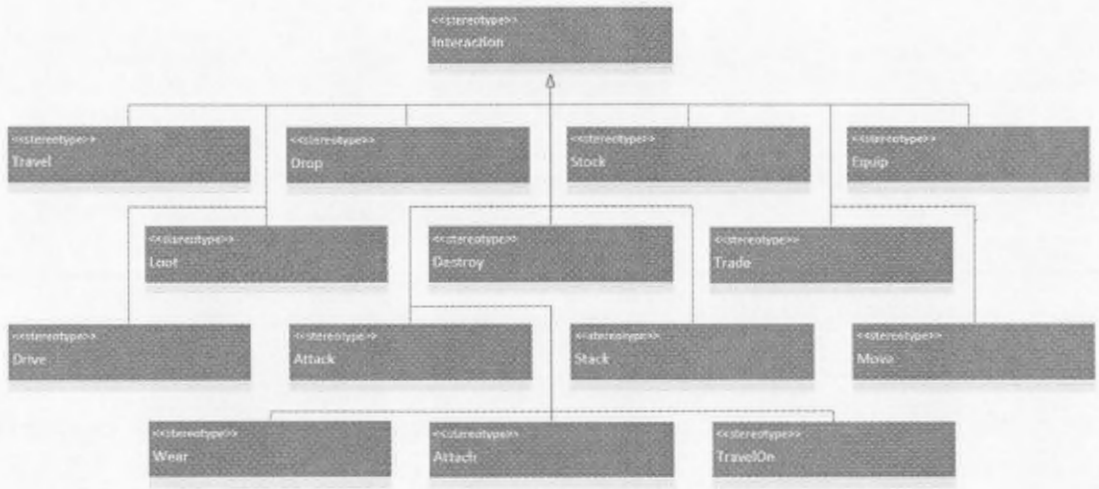


Figure A.18: Interaction

# BIBLIOGRAPHIE


[1] Correa, P. et Möller, L. (2014). Improving game development process applying multi-view game design documents. https://www.researchgate.net/publication/280735266_Improving_Game_Development_Process_Applying_Multi-View_Game_Design_Documents.

[2] Duarte, L. C. S. (02/03/2015). Revisiting the MDA framework. https://www.gamasutra.com/blogs/LuizClaudioSilveiraDuarte/20150203/233487/Revisiting_the_MDA_framework.php.

[3] Freeman, T. (1997). Creating a great design document. https://www.gamasutra.com/view/feature/131632/creating_a_great_design_document.php.

[4] Heintz, S. et Law, E. (2015). The game genre map : A revised game classification. Dans *Proceedings of the 2015 Annual Symposium on computer-human interaction in play*, CHI PLAY '15, 175–184. ACM.

[5] Hunicke, R., Leblanc, M. et Zubek, R. (2004). MDA : A formal approach to game design and game research. *AAAI Workshop - Technical Report, 1.*

[6] III, R. R. (2010). *Game Design : Theory & Practice (2nd edition).* Wordware Publishing, Inc.

[7] Librande, S. (2010). One page designs. Game Developers Conference.

[8] Machado, T. L. D. A., Ramalho, G. L., Alves, C. F., Garcia, V. C., Araujo, L. F., Lemos, V., Gomes, V. C. F., Silva, A. P., Barros, X. S., Centro, E. S.

A. R. et Sistemas, E. D. (2010). Game development guidelines : Practices to avoid conflicts between software and design.

[9] Marais, C., Futcher, L. et van Niekerk, J. (2014). Incorporating a game design document into game development projects deliverables. SACLA 2014.

[10] OMG (2017). Unified Modeling Language. `https://www.omg.org/spec/UML/2.5.1/PDF`.

[11] Parmentier, G. et Mangematin, V. (2009). Innovation et création dans le jeu vidéo. *Revue française de gestion*, *191*(1), 71–87.

[12] Pedersen, R. E. (2003). *Game Design Foundations.* Wordware Publishing, Inc.

[13] Rogers, S. (2014). *Level UP, The Guide to Greate Video Game Design (2nd Edition).*

[14] Rollings, A. et Morris, D. (2004). *Game Architecture and Design :.* `http://dx.doi.org/10.1016/j.joca.2015.06.008`

[15] Salazar, M. G., Mitre, H. A., Olalde, C. L. et Sánchez, J. L. G. (2012). Proposal of game design document from software engineering requirements perspective. Dans *Proceedings of CGAMES'2012 USA - 17th International Conference on Computer Games : AI, Animation, Mobile, Interactive Multimedia, Educational and Serious Games*, 81–85. CGAMES'2012 USA.

[16] Salazar, M. G., Mitre, H. A., Olalde, C. L. et Sánchez, J. L. G. (2019). GDD Template based on the Donkey Kong game. `https://www.cimat.mx/~hmitre/GDD-DKS.pdf`.

[17] Salen, K. et Zimmerman, E. (2013). Rules of Play : Game design fundamentals. *International Immunology*, *25*(8), NP–NP. http://dx.doi.org/10.1093/intimm/dxs150

[18] Takami, K. et Oniki, Y. (2003). *Battle Royale.* VIZ, LLC. Récupéré de https://books.google.ca/books?id=1fB8p-dZLSIC

[19] Walk, W., Görlich, D. et Barrett, M. (2017). *Design, Dynamics, Experience (DDE) : An Advancement of the MDA Framework for Game Design*, (p. 27–45).

[20] Warmer, J. et Kleppe, A. (2003). *The Object Constraint Language (2nd Edition) : Getting Your Models Ready for MDA.* Addison-Wesley.

[21] Winn, B. M. (2011). The Design, Play, and Experience Framework. *Handbook of Research on Effective Electronic Gaming in Education*, *5497*, 1010–1024. http://dx.doi.org/10.4018/978-1-59904-808-6.ch058