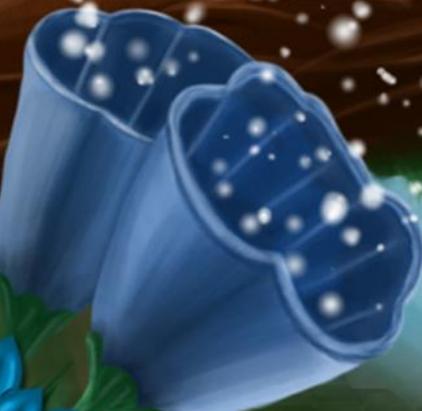


Bouhadida Achref & Neumann Anas,
Encadrés par Madame Malek Jihene.



the *Twi*ngs

Rapport de projet de fin d'études pour l'obtention de la licence
Informatique et Multimédia



Année universitaire 2013 – 2014

Dédicaces

Anas Neumann

« Je dédicace ce projet à tous les membres de l'association du GDA Sidi Amor, ma famille et tout particulièrement à ma mère. »

Achref Bouhadida

« Je tiens absolument à remercier toutes les personnes qui ont participé à la réalisation de ce projet de fin d'études, que ce soit d'une façon directe ou indirecte.

Cette aventure était passionnante mais aussi très hardie. La voie que nous avions choisie de suivre était truffée de pièges mais notre encadrante, Jihen Malek, a toujours été présente, avec nous, avec ses connaissances et sa sagesse pour assurer un bon avancement des événements : un mentor digne de son rang. Je tiens donc spécialement à la remercier du fond de la chose qui anime mon corps qu'est mon cœur.

La deuxième personne à remercier est tout simplement mon collègue mais surtout, avant tout, un meilleur et un cher ami, Anas Neumann, mon compagnon de cette aventure. Un brave guerrier qui sait mettre les coups là où il faut et qui sait aussi encaisser les plus lourds des dégâts. Son courage sans failles nous a permis de nous évader des plus profonds et dangereux donjons, sa maîtrise des armes a éradiqué tous les obstacles, sans exceptions.

StolenPad Studio, un merveilleux village. Ses gens offrent le meilleur des services sans rien demander en retour. Haroun Gharbi, le chef, nous a ouvert les bras et nous a accueillis chaleureusement comme si nous étions les uns de ses membres. Nadhir Zaeim, Hichem Zarrad, Mostapaha Dhaouadi et finalement Ahmed Chlagou nous ont offert l'accès à leurs bibliothèques en nous montrant tous les secrets du domaine et en nous apprenant toutes les compétences nécessaires à l'achèvement de notre quête. Un merci titanique à tous les membres de StolenPad, ils ont été géniaux.

Des prières pleines d'amour à notre Cher Unique Dieu qui m'a tout simplement tout fourni. Il m'a fait entourer d'une famille, qui ne connaît pas de limites pour des notions de gentillesse ou de charité. Une mère faite de tendresse, un père fait d'honneur, j'ai été entouré par la lumière des anges qui ont veillé sur moi tout au long de ce périple. Eux, je leur consacre mon âme. »

Remerciements

Nous tenions à remercier le corps administratif et enseignant de l' « **Institut Supérieur des Arts Multimédia de Manouba** » pour les divers enseignements qui nous ont permis la création de The Twins.

Parmi ces enseignants, nous tenions à remercier plus particulièrement Madame **Jihène Malek**, notre encadrante. Elle a sacrifié son temps pour nous conseiller et nous corriger en matière de conduite de projet. Les richesses de ses idées ont introduit dans ce simple projet étudiant un goût de professionnalisme.

Nous souhaitons ensuite à remercier l'équipe de « **StolenPad Studio** » au complet. Nous commencerons par Monsieur **Haroun Gharbi** qui nous a acceptés volontiers sachant que nous n'étions encore que débutants. Nous remercions les développeurs et particulièrement **Ahmed Chlagou** qui n'a pas hésité à partager ses connaissances du moteur Unity. Nous remercions **Nadhir Zaeim** et **Hichem Zarrad** pour le savoir qu'ils nous ont transmis. À l'écoute à tout moment et même en dehors des heures de travail, ils ont complété nos manques en matière de « Game Art » et conseillé sur les meilleures méthodes pour obtenir facilement un beau résultat. **Moustapha Dhaouadi**. Spécialiste dans l'animation 2D et 3D, pour sa disponibilité et l'intérêt qu'il a porté à notre projet.

Ensuite, beaucoup de personnes ont participé activement, en testant ou donnant leurs avis, et d'autres simplement en nous soutenant moralement. Ainsi, nous tenions à remercier toutes les personnes, de nos **familles** et de nos **amis**, qui nous ont soutenus tout au long de ce semestre de travail.

Sommaire

Introduction	6
Contexte général du projet.....	7
Organisme d'accueil.....	7
Description du sujet.....	7
Approche méthodologique	8
Organisation du rapport	10
État de l'Art.....	12
1. Le monde des jeux vidéo.....	13
2. Étude des jeux similaires	22
3. Les sites web, un outil de distribution.....	36
4. Étude des sites web similaires	39
The Twins	41
1. Synopsis de l'application « The Twins ».....	42
2. Spécifications fonctionnelles.....	44
3. Conception de l'application.....	49
4. Contraintes de qualité	57
L'aventure commence	63
1. Déroulement du projet et méthode de travail	64
2. Backlog du produit et plan des releases.....	68
Système de combat multi-joueurs et connexion en ligne	73
1. Tableau prévisionnel du sprint	73
2. Création des personnages du jeu	76
3. Animation des personnages	92
4. Création des pouvoirs.....	93
5. Intégration des personnages	94
6. Programmation des personnages.....	99
7. Un jeu multi-joueurs.....	112
8. Unity et les bases de données.....	114
L'aventure continue	117
Création de l'environnement et des énigmes	118
1. Tableau prévisionnel du sprint	118

2. Conception et création graphique.....	120
3. Composition de l'environnement.....	125
4. Développement des énigmes et du temps.....	128
Création des menus et autres interfaces utilisateurs.....	134
1. Tableau prévisionnel du sprint	134
2. Création graphique des éléments d'interfaces.....	135
3. Développement des interfaces.....	143
L'aventure s'achève.....	148
Création du site web et des autres supports.....	149
1. Tableau prévisionnel du sprint	149
2. Présentation du site web.....	150
3. Création graphique	150
4. Développement des pages du site	155
5. Hébergement du site web	160
6. Plusieurs supports de distribution.....	161
Finalisation du projet.....	165
1. Finalisation et ajout d'effets visuels et sonores	165
2. Exportation de l'exécutable	169
Conclusion.....	173
1. Bilan du projet	174
2. Bilan personnel.....	175
Table des figures.....	176
Table des tableaux.....	181
Ludographie et Bibliographie	182



Introduction

Contexte général du projet

En tant qu'étudiants en troisième année de la licence Informatique et Multimédia à l'Institut Supérieur des Arts Multimédias de la Manouba (ISAMM), nous avons réalisé une application multimédia dans le cadre de notre projet de fin d'études.

Notre groupe (référence : IM7), constitué d'Achref Bouhadida et Anas Neumann, encadrés par Madame Jihène Malek, a choisi de développer un jeu vidéo tridimensionnel, « **The Twins** »¹, ainsi que les différents supports de distribution tels que le site web officiel du jeu, la page Facebook de notre groupe de travail que l'on a nommé « Prod'IT Studio », ou encore la chaîne Youtube de « Prod'IT Studio ».

La création de « **The Twins** » et de l'ensemble de notre projet impliqua une grande diversité de tâches. Nous avons coopéré sur ces différentes parties du travail pour lui donner vie.

Organisme d'accueil

Après avoir contacté l'entreprise « **StolenPad Studio** » et avoir passé un entretien, nous avons été acceptés en tant que stagiaires, encadrés par l'équipe du studio, pour la réalisation d'un jeu pour ordinateur personnel (PC). Les différents membres du studio nous ont ainsi conseillés sur les tâches de notre jeu liées à leurs spécialités respectives au sein de StolenPad.

StolenPad Studio 3D Works est un studio de développement d'applications multimédias (sites web, images de synthèse, publicités interactives...) dont la principale activité est le jeu 3D et en particulier pour téléphone mobile. Le studio est une entreprise indépendante dont le local se situe à Menzah 9 (dans le gouvernorat de l'Ariana). La société est formée d'une seule équipe de onze personnes dont deux développeurs, un artiste de conception, un modélisateur 3D, un animateur 2D/3D, un ingénieur du son, un directeur financier et le chef du studio qui en est également le fondateur.

Le premier jeu développé par StolenPad, « **Tin&Food** », est en développement depuis maintenant deux ans et sera, après une dernière phase de finalisation, distribué via le « Google Play » et donc disponible pour tout utilisateur propriétaire d'un mobile supportant le système « Android ».

Description du sujet

Le principal objectif de notre projet fut de concevoir et développer un jeu destiné aux enfants. Il nous a donc fallu choisir un type d'univers haut en couleurs, agréable à regarder, peuplé de personnages comiques. Nous avons également visé une utilisation instinctive du produit qui malgré cela offrirait une expérience complète intéressante aux joueurs aussi bien en terme de fonctionnalités, de scénario que de l'ambiance créée. Pour cela nous avons choisi d'orienter le principe du jeu dans la découverte, la résolution d'énigmes adaptées aux enfants et l'aventure.

Le déroulement du jeu se fait en « **side-scrolling** »² donc une vue de profil et un déroulement de l'univers sur deux axes. **The Twins** est à la fois un jeu de **plateforme** (ce qui inclut le principe du saut et de l'action), un jeu de **réflexion** (ce qui signifie la présence d'énigmes) et un **RPG**³.

¹ Le nom de notre application multimédia.

² Défilement sur le côté, comme un parchemin.

En cohésion avec tout ceci, l'idée de faire un jeu pour enfants a fait intervenir dans le projet les idées du vidéo-ludique éducatif. Nous avons donc voulu que notre jeu apporte une utilité dans ce sens. Ainsi nous avons décidé, pour réduire le phénomène d'isolement et de renfermement grandissant chez les enfants jouant excessivement aux jeux vidéo, que le jeu ne pourrait être joué qu'à deux. Nous avons favorisé d'une part la proximité physique des deux enfants et pour cela le mode de jeu est en réseau local et donc limité géographiquement, et d'autre part la proximité morale en les faisant jouer ensemble et non l'un contre l'autre. En effet, les deux joueurs se voient contraints dans le jeu de s'entre-aider mutuellement pour parvenir à en finir l'histoire et venir à bout des pièges et autres embûches dont le monde est parsemé.

Approche méthodologique

Les deux membres de notre équipe étant passionnés de jeux vidéo depuis l'enfance et décidés à travailler dans le domaine depuis plusieurs années déjà, nous avons voulu atteindre de la manière la plus proche possible le résultat d'un jeu professionnel. Nous souhaitons que le groupe crée, Prod'IT Studio, ainsi que **The Twins** ne soient qu'un début et que le jeu réussisse à insuffler cette sensation qui a bercé notre enfance.

The Twins au carrefour des disciplines :

En respectant cela, il fallait également considérer l'intérêt pédagogique du projet en tant que qu'un travail scolaire. Ainsi nous avons lié les deux objectifs en introduisant dans notre travail le plus grand nombre de matières étudiées en cours qui seraient pertinentes et utiles dans la réalisation du projet. Ces trois années à l'ISAMM nous ont permis d'accéder aux notions qui ont fait de **The Twins** une étude scientifique poussée à travers l'utilisation des connaissances acquises dans les matières suivantes :

- *Réseaux Multimédias* pour les notions nécessaires au mode multi-joueurs (Adresse IP, serveur...),
- *Intelligence Artificielle* dans la création des modes de réflexion des ennemis contrôlés automatiquement par l'ordinateur,
- *Animation 3D* dans la création des objets, des personnages et de leurs animations,
- *Unity* pour l'utilisation du moteur Unity 3D,
- *Sécurité des Données Multimédias* pour chiffrer les mots de passe dans notre base de données,
- *Bases de Données* pour la conception et la création de notre base de données de scores,
- *PHP* dans la réalisation du site web du jeu ainsi que dans l'interaction entre le jeu et la base de données hébergée en ligne,
- *Conception Orientée Objets et Génie Logiciel* pour parfaire le processus de développement du projet et réaliser les phases d'analyse et de conception du projet,
- *Programmation Orientée Objet* afin de programmer d'une manière efficace, réutilisable et optimisée,
- *Design et Conception Graphique* dans l'utilisation des logiciels de la collection Adobe (Photoshop et Illustrator) pour la conception des personnages et objets 3D, la peinture des

³ Role Playing Game : jeu de rôle dans lequel chaque personnage a une fonction précise et unique et dans lequel les modes de combat sont fortement inspirés du principe du tour par tour.

textures (sur UVs) des modélisations 3D, la peinture des « SpriteSheets »⁴ ainsi que pour la création des interfaces utilisateur,

- *Architecture Logiciels* utilisée pour garantir une maintenance facile et une future évolution à notre jeu en organisant de manière logique tous les composants de notre projet.

Approches adoptées pour l'ingénierie du jeu :

Nous avons choisi d'utiliser tout au long des quatre mois de travail :

- Le langage modélisation **UML** (Unified Modeling Language) pour la spécification des besoins et la conception des différentes parties du projet.
- Le modèle de cycle de vie **Scrum** comme processus de travail garantissant une flexibilité dans la gestion du projet qui avait, étant débutant, des spécifications qui risquaient de changer à tout moment. SCRUM est également un processus centré sur l'équipe de travail ce qui correspond très bien à la situation d'un projet étudiant dont la finalité n'est pas tant le produit mais également le travail fourni par les membres qui travaillent dessus. Le cycle de vie était donc incrémental en respectant les spécialités des incrémentés spécifiques à SCRUM : les « Sprints ».
- Le principe, inspiré de **XP** (eXtreme Programming), de développer toujours à deux pour nous garantir de trouver la meilleure solution à un problème donné.

Répartition du travail

Notre groupe étant composé des deux étudiants, il convenait de diviser le travail en deux parties et d'assigner à chacun d'entre nous l'une d'entre elles. Afin d'effectuer cette séparation de manière convenable, il nous fallait d'abord étudier comment elle se fait dans le monde professionnel.

Bien qu'au début, le monde du jeu vidéo était assez informel et les métiers qui le comptaient étaient mal définis, aujourd'hui l'industrie du jeu vidéo est devenue une source de production de richesse considérable notamment dans certains pays tels que le Canada qui possède la plus grande équipe du studio français Ubisoft. Les entreprises spécialisées dans ce domaine comptent parmi elles de nombreuses véritables multinationales telles qu'Ubisoft, Electronic Arts, Sony, Atari, Blizzard, Valve et d'autres qui travaillent surtout sur des jeux vidéo destinés aux consoles et aux ordinateurs mais aussi un nombre incalculable de petits studios indépendants travaillant principalement sur des projets pour téléphones mobiles tels que Imangi Studio (Temple Run), Half Brick Studio (Fruit Ninja, JetPack Joyride) ou encore StolenPad Studio (Tin&Food) ; l'entreprise tunisienne qui nous a encadrés dans ce projet. Dans ces entreprises, les métiers du jeu sont devenus normalisés selon des nomenclatures mondiales et nous pouvons les résumer grâce à un schéma simplifié [voir figure 1] (<http://www.millenium.org>).

⁴ Document qui contient toutes les étapes successives en images formant l'affichage d'une animation 2D.

Les métiers du Jeu vidéo



« Figure 1 : les métiers du jeu vidéo »

Ainsi, Anas Neumann s'appliqua principalement sur celles liées à la programmation et à l'utilisation du moteur Unity 3D alors qu'Achref Bouhadida s'occupa en majorité des tâches en rapport avec le « **Game Art** »⁵ et le « **Game Design** »⁶.

Logiciels utilisés

Nous comptions également utiliser les logiciels suivants :

- *Unity 3D* version 4.0 en tant que moteur graphique et physique du jeu,
- *Autodesk 3DS Max* et *Maya* pour la modélisation, le découpage des UVs et l'animation 3D,
- *Adobe Photoshop* pour la conception graphique et le texturing,
- *NotePad++* pour le développement PHP et SQL,
- *FileZilla* pour l'envoi en ligne (FTP) et le déploiement des scripts PHP et de l'ensemble des composants du site web,
- *Monodevelop* comme plateforme de programmation pour le jeu (UnityScript),
- *Trello*, application web utilisée pour la tenue du tableau de tâches de Scrum et pour l'échange de fichiers avec Madame Jihène Malek.

Organisation du rapport

La description de la création d'un jeu vidéo réalisé sous forme d'un projet scolaire peut se révéler assez délicate. Dans l'idée de tout présenter en garantissant une organisation logique à la lecture et une facilité de suivie, nous avons décidé de raconter cette aventure en partie dans l'ordre chronologique des événements. Ainsi, après avoir exposé nos choix et influences et après avoir détaillé le synopsis du jeu et les fonctionnalités qui le composeront, nous commencerons à décrire les étapes de ces quatre mois telles qu'elles se sont produites. Évidemment, nous respecterons en

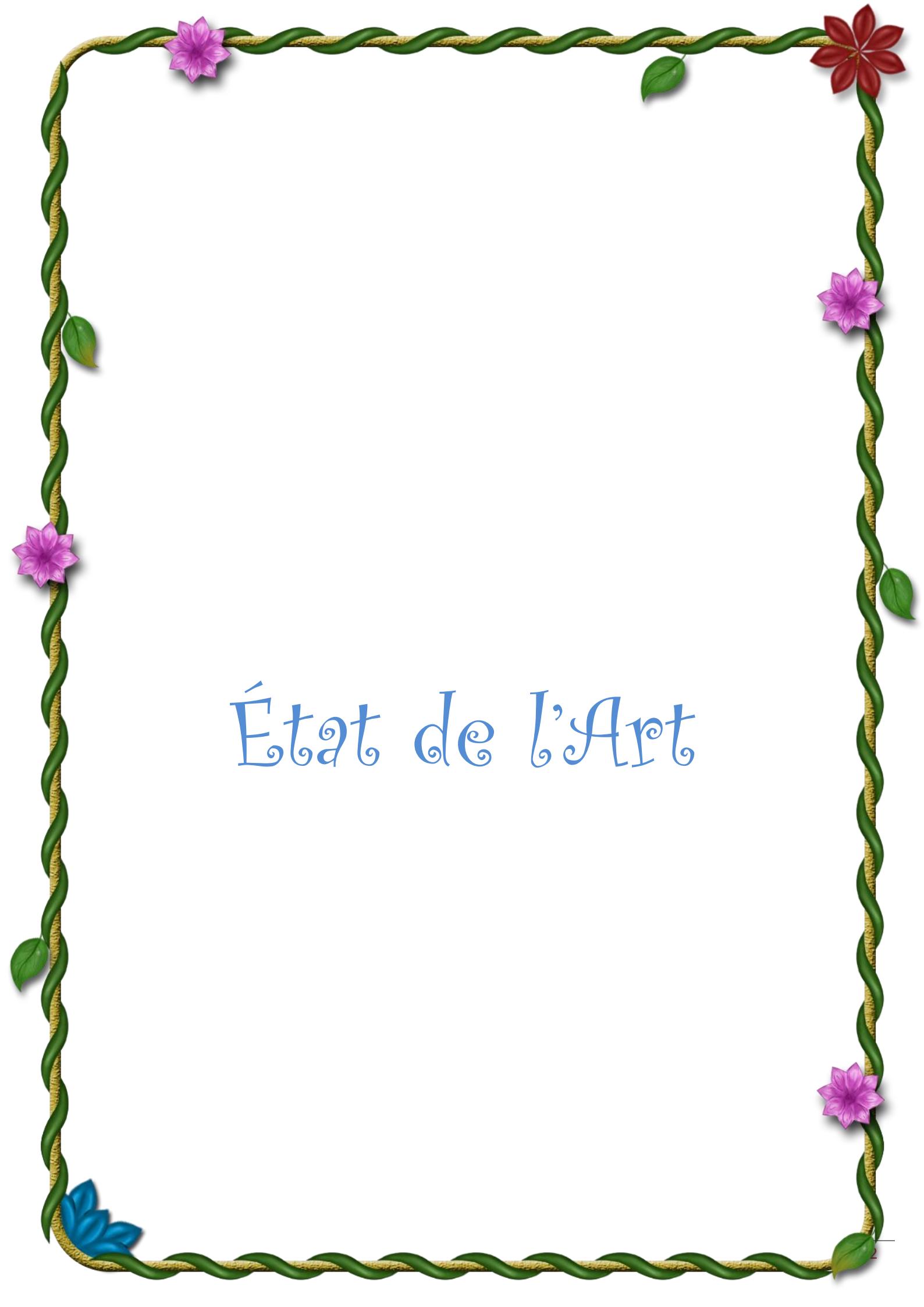
⁵ L'art du jeu : peinture digitale, conception graphique, sculpture digitale, etc...

⁶ Le design du jeu : design d'interfaces, level design, etc...

parallèle à ceci les règles d'organisation d'un rapport et essaierons de lier l'agréable au professionnalisme.

Les trois grandes parties « *L'aventure commence* », « *L'aventure se poursuit* » et « *L'aventure s'achève* » correspondent à cette partie racontée fidèlement à l'histoire. Elles sont découpées en sprints (parties du projet ou incrément) dans l'ordre dans lequel ils ont été réalisés. Chaque sprint a occupé entre deux et quatre semaines du projet et possède toutes les phases du cycle de vie d'un logiciel.

Cependant, avant cela, nous allons commencer par vous présenter brièvement le monde du jeu vidéo tridimensionnel pour en sortir les bases d'influence dont nous nous sommes inspirés pour l'imagination de **The Twins**.



État de l'Art

Avant d'être capable de prétendre pouvoir développer un jeu vidéo, il est nécessaire d'avoir une vision globale du domaine. Pour cela nous allons vous présenter le monde des jeux vidéo et essayerons d'en obtenir suffisamment de connaissances et d'inspiration pour The Twins.

1. Le monde des jeux vidéo

1.1. Les types de jeux vidéo

On attribut généralement le nom de « jeu vidéo » aux applications de type multimédia dont le but est souvent la détente de l'utilisateur et qui sont jouées sur des supports matériels tels que les consoles de jeux, les ordinateurs personnels, les téléphones mobiles et les bornes d'arcades. À mi-chemin entre l'art et l'industrie, le jeu vidéo est un sous-domaine jeu électronique ayant la spécificité d'être affiché sur un écran et dans lequel le joueur interagit dynamiquement en utilisant une interface utilisateur. Bien qu'il n'existe pas une date parfaite de l'apparition du domaine, on considère qu'il serait né dans les années 1950 grâce aux inventions de l'allemand Ralph Baer (www.wikipedia.org).

Il existe plusieurs manières de différencier les types de jeux vidéo, on peut par exemple les classer selon le thème ou l'univers présent dans le jeu ou encore par le type de « **gameplay** »⁷. Cependant, il existe globalement deux types de jeu et un type intermédiaire : Les jeux 2D, Les jeux vidéo 3D (3D) et les jeux 2.5.

1.1.1 Les jeux en deux dimensions (2D)

Les jeux en deux dimensions, qui sont les premiers apparus car plus légers graphiquement et en calcul pour l'ordinateur. Dans un jeu 2D, les actions, déplacements et affichages se font uniquement sur deux axes (hauteurs et largeur). Divers et variés, les jeux 2D ont beaucoup évolués depuis leur apparition. Ils naquirent avec des jeux tels que le « Tetris » ou le « casse-brique » qui sont devenus aujourd'hui des références classiques ayant inspiré les jeux plus évolués venus par la suite. Aujourd'hui des jeux 2D de hautes qualités continuent d'apparaître tels que « Child Of Light » (Ubisoft, 2014) [voir figure 2] du studio français / canadien Ubisoft.

⁷ Manière de jouer et fonctionnalités présente dans le jeu.



« Figure 2 : *Child of Light* »

Cependant pour contrer le préjugé du jeu « **old-school** »⁸ et démodé, les studios qui créent des jeux 2D aujourd'hui, favorisent l'aspect artistique et l'originalité de leurs projets afin d'apporter un avantage face aux jeux plus modernes. On peut citer comme exemple à cela le chef d'œuvre « Limbo » ([Playdead, 2011](#)) [voir figure 3].



« Figure 3 : *Limbo* »

1.1.2. Les jeux vidéo tridimensionnels (3D)

- Les jeux vidéo tridimensionnels (3D) qui, apparus quelques années plus tard avec l'évolution des supports matériels, offrent l'axe de la profondeur en plus des deux premiers axes

⁸ Jeux des anciennes générations.

(hauteur et largeur). L'apparition de la troisième dimension dans le jeu accorda de nombreuses possibilités aux développeurs de jeux qui inventèrent très vite de nouveaux types de jeux inaccessibles jusque là. Tout comme les jeux 2D, les jeux tridimensionnels ont eux aussi évolué depuis leur création et cela se remarque du point de vue du joueur principalement sur le plan graphique. Nous pouvons faire la comparaison assez extrême entre « Quake 3 » ([id Software, 1999](#)) [voir figure 4] l'un des premiers jeux tridimensionnels créés et « inFAMOUS : Second Son » ([SuckerPunch, 2014](#)) [voir figure 5] exclusivité PS4⁹, l'un des jeux les plus modernes qui soient.



« Figure 4 : Quake 3 »

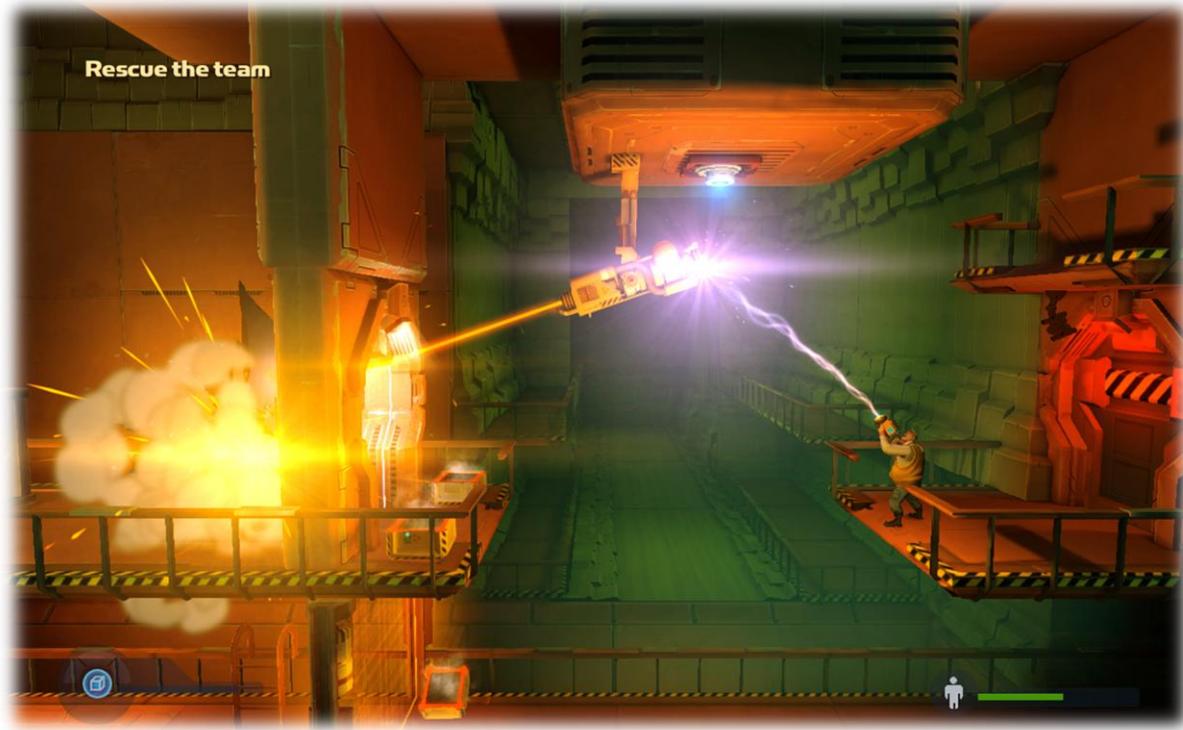


« Figure 5 : inFAMOUS : Second Son »

⁹ Playstation 4, la console de salon de Sony.

1.1.3. Les jeux en 2.5D

- Enfin, il existe comme nous l'avons dit un type intermédiaire appelé communément le type « **2.5D** ». C'est un type de jeu dans lesquels les graphismes du jeu sont réalisés en 3D (les modélisations) et le gameplay (déplacements et actions) reste en 2D ou bien le contraire. Nous pouvons citer comme exemple de jeu 2.5D réalisé à l'aide du moteur physique Unity 3D : « *Rochard* » ([Recoil Games, 2011](#)) [voir figure 6].



« Figure 6 : *Rochard* »

1.2 Les différents gameplays et univers

Chaque jeu vidéo annoncé par un studio reconnu se voit, avant sa sortie officielle, attribuer une valeur « **PEGI** »¹⁰ qui détermine l'âge à partir duquel un joueur peut y jouer et est également jugé par des instituts tels que « **IGN** » (américain) ou « **jeuxvideo.com** » (européen) qui lui donne une note sur vingt points. Un joueur peut ainsi se faire une idée du jeu et se décider s'il va l'acheter ou non sans prendre le risque d'être déçu. Les deux instituts cités précédemment attribuent aussi un type aux jeux annoncés en fonction de leurs univers (les thèmes de mondes et de personnages) et de leur gameplay. Au fil des années, tout comme les courants musicaux ou les époques de peintures, les types de jeux se multiplient. Une nouvelle catégorie née en se basant sur une autre plus ancienne ou au contraire en s'opposant aux idées d'une autre et pourra donner par la suite naissance à de futures sous-catégories. Pour présenter ces différents gameplays et univers sans détailler excessivement, nous définirons uniquement les grandes catégories référencées par l'article qui concerne les jeux vidéo sur « **Wikipedia** ».

¹⁰ Pan European Game Information.

1.2.1. Les types de gameplays

- « *Combat* », type dans lequel le personnage contrôlé par le joueur est en danger et doit se battre. Nous pouvons citer, comme référence dans ce type, la saga « Street Fighter » ([Capcom, 2013](#)),
- « *Tir* », type dans lequel le personnage possède une arme et le joueur doit se servir de son habileté à viser avec vitesse et précision. « Resogun » ([Sony, 2013](#)) est un parfait exemple de jeu de tir,
- « *Plateforme* », dont le plus connu est sans nul doute « Mario Brothers » ([Nintendo, 2013](#)), Dans un jeu de plateforme, le joueur se trouve dans un univers instable où il doit utiliser ses réflexes en termes de rapidité pour sauter et esquiver de nombreux pièges,
- « *Aventure* », type utilisé afin que le joueur sorte du monde dans lequel il vit tous les jours pour se plonger dans un autre plus mystérieux qu'il va devoir explorer. Nous pouvons citer en tant qu'exemple clé le célèbre « Gone Home » ([The Fullbright Company, 2013](#)),
- « *Action-aventure* », est un type de jeu à mi-chemin entre la découverte / exploration et le combat. Un jeu de ce type, la saga « Tomb Raider » ([Square Enix, 2013](#)), a fait un tel succès qu'elle fut par la suite adaptée en film hollywoodien,
- « *Jeu de rôle* », type connu pour être complexe et addictif. Dans un jeu de rôle, le joueur doit choisir quel sera, par exemple, la race et les caractéristiques de son personnage. Personnage dont il va ensuite vivre la vie pour le faire évoluer. Le style est connu pour être addictif car le joueur peut finir par s'identifier à son personnage. « Final Fantasy Online : Realm Reborn » ([Square Enix, 2013](#)) ou « World Of Warcraft » ([Blizzard, 2005](#)) sont deux titres du domaine qui connaissent une fidélité des joueurs depuis plusieurs années déjà,
- « *Réflexion* » type dans lequel le joueur peut se trouver bloqué face à des choix à multiples solutions ou encore face à une énigme dont la solution est cachée. « Limbo » est un jeu dans lequel le joueur n'est averti de rien et doit constamment réfléchir à ce qu'il doit faire pour avancer dans le jeu,
- « *Simulation* », les jeux de simulation servent à essayer de reproduire la sensation produite dans une situation réelle telle que la conduite d'un avion, d'un camion ou encore d'un bateau. Le jeu « Flight Simulator » ([Microsoft, 2006](#)) se concentre sur la simulation de conduite d'un avion,
- « *Stratégie* » type dans lequel la capacité d'un joueur à gérer des ressources, évaluer des risques et commander une armée est mise en valeur. La saga « Age of Empire » ([Ensemble Studios, 2005](#)) fait partie de ce type de jeu et fut copiée de nombreuses fois par d'autres studios,
- « *Sport* ». Dans la même idée que les jeux de simulation, le jeu de sport reproduit la sensation d'une activité existante mais ne se concentre pas sur la conduite d'un véhicule mais plutôt sur la reproduction d'un match sportif. « FIFA »¹¹ ([Electronic Arts, 2013](#)) est l'un des jeux de football les plus connus avec son concurrent « PES » ([Konami, 2013](#)),
- « *Rythme* » comme par exemple dans le jeu très apprécié « Guitar Hero : World Tour » ([Neversoft, 2009](#)), le principe des jeux de rythme est de tester la capacité d'un joueur à rester synchronisé avec un rythme et une vitesse déterminée par le jeu.
- « *Course* », jeu dans lequel le joueur doit atteindre un objectif le plus vite possible avant ses adversaires ou, si il n'y en a pas, avant une limite de temps donnée.

¹¹ Federation International of Football Association.

1.2.2 Les types d'univers

- L'univers « **Fantastique** » est un univers basé sur l'époque moyenâgeuse mais avec une touche de magie et de créatures telles que les elfes, les géants ou les dragons. Le fantastique existait dans les romans bien avant d'apparaître dans le jeu vidéo (« Lord Of The Ring » de J.R.R Tolkien). Les touches magiques du fantastique proviennent des légendes germaniques (gobelins, etc...), des traditions asiatiques (dragon, sabres ...) et des différentes mythologies. On peut citer dans cet univers les jeux « Guild Wars » ([ArenaNet, 2012](#)) [voir figure 7], un MMORPG¹² ou « Battle For The Middle Earth II» ([Electronic Arts, 2006](#)) un RTS¹³.



« Figure 7 : Guild Wars 2 »

- L'univers « **Futuriste** » est un univers tiré de la science fiction. Tout comme le fantastique, l'univers était déjà présent dans les romans bien avant de l'être dans le jeu vidéo. L'histoire s'y déroule dans un futur où la technologie a évolué par rapport à notre époque et où des créatures venues d'autres planètes ont pris contact avec notre civilisation. On peut ici citer « KillZone : Shadow Fall » ([Guerrilla, 2013](#)) [voir figure 8] un FPS¹⁴.

¹² Massively Multiplayer Online Role Playing Game : jeu de rôle massivement multijoueurs.

¹³ Real Time Stratégie : stratégie en temps réel.

¹⁴ First Person Shooter : jeu de tir à la première personne.



« Figure 8 : KillZone : Shadow Fall »

- Les univers « **Modernes** » et « **Urbains** ». Dans ces jeux, l'histoire se déroule dans une époque proche de la notre et dans un environnement tel que nous pouvons en voir dans la vraie vie. Ainsi le joueur a l'impression plus ou moins forte de ne pas être dans un jeu mais de pouvoir faire des choses qui lui sont impossibles normalement. On peut citer ici « GTA : San Andreas »¹⁵ (Rockstar, 2004) [voir figure 9].



« Figure 9 : Grand Thief Auto : San Andreas »

¹⁵ Grand Thief Auto, saga de Rockstar : <http://www.rockstargames.com>.

1.3. Les modes de jeux vidéo

Il existe encore une autre manière de classer les jeux vidéo en deux catégories distinctes : le mode de jeu. On peut alors considérer qu'il existe d'un côté les jeux en mode « solo » où le joueur est seul dans sa partie et d'un autre côté les jeux en mode « multi-joueurs » où l'on peut voir plusieurs joueurs sur une seule partie. Cependant, cette méthode de classification a ses limites car un grand nombre de jeux possède à la fois un mode solo appelé souvent « mode campagne » ou « mode histoire » et un mode multi-joueurs. D'autres jeux, eux, ne se concentre uniquement que sur le multi-joueurs, on les appelle les MMOG¹⁶. De plus, il existe aussi des différences au sein du groupe des jeux multi-joueurs. En effet, dans certains cas, les joueurs jouent en coopération (PVE¹⁷), dans d'autres, ils jouent les uns contre les autres (PVP¹⁸) ou encore les deux ensemble. En outre, que ce soit en PVP ou même en PVE, il existe encore une différence attachée au matériel utilisé. Par exemple certains jeux multi-joueurs se jouent sur une seule machine (pc, console...) et dans ce cas, l'écran peut alors être splitté¹⁹ en plusieurs parties. Dans d'autres jeux, chaque joueur possède sa machine et les joueurs jouent donc soit en ligne à travers un serveur appartenant au studio qui a créé le jeu soit en réseau local et dans ce cas, l'un des joueurs est le serveur. Ainsi, c'est un moyen de classification délicat et par conséquent peu approprié.

Généralement, le mode multi-joueurs offre plus de défis que le mode solo car chaque joueur doit se mesurer à l'intelligence et aux réflexes d'un autre joueur qui peut deviner ses intentions. De plus, contrairement au mode « Histoire » en solo dans lequel le jeu se termine après avoir fini une histoire, le mode multi-joueurs ne se termine jamais. Ainsi, beaucoup de joueurs préfèrent le mode multi-joueurs et la combinaison des deux modes pour avoir une durée de vie de jeu plus longue.

Certains jeux comme ceux de type combat, de course ou encore de type jeu de société tels que « Mortal Kombat » ([NetherRealm Studios, 2011](#)) ou « Playstation All-stars Battle Royale » ([SuperBot, 2012](#)) ayant un mode multi-joueurs sont également très appréciés par les joueurs car ils sont en quelques sortes les héritiers du jeu de société sur plateau. Avant, on jouait entre amis dans le salon au « Monopoly », maintenant on peut jouer en famille [voir figure 10] sur une console comme la Playstation.

¹⁶ Massively Multiplayer Online Games : jeux massivement multijoueurs.

¹⁷ Players Versus Environment : joueurs contre l'environnement (intelligence artificielle).

¹⁸ Players Versus Players : joueurs contre joueurs (jeu compétitif).

¹⁹ Écran divisé en deux parties, soit verticalement soit horizontalement.



« Figure 10 : une famille jouant à un jeu multi-joueurs »

1.4. Bilan et choix

Pour la création de notre jeu, **The Twins**, nous avons donc du faire des choix d'univers, de types de jeu et de modes de jeu et ces choix devaient être en accord avec le fait que le jeu serait pour enfants.

- Ainsi nous avons jugé que pour obtenir le monde **coloré** et **agréable** que nous souhaitions, l'univers « **Fantastique** » était celui qui correspondrait le mieux. Il offrait la possibilité de créer un monde forestier plein de fleurs de toutes les couleurs ou encore de créer des attaques de type magique et donc avec des formes et des couleurs qui peuvent ne pas exister dans la réelle nature sans que cela soit gênant.
- Nous avons ensuite choisi de combiner plusieurs types parmi les gameplays cités précédemment :
 - Le type jeu « **Plateformes** », donc nos personnages pourront sauter, se battre et récupérer des objets sur leur trajet.
 - Le type jeu « **Réflexion** », ainsi nos personnages devront parfois s'arrêter pour réfléchir au moyen de passer un obstacle.
 - Le type « **Jeu de Rôle** », et donc chacun de nos personnages aura ses capacités personnelles. Généralement dans les jeux de rôle, les capacités des personnages varient en fonction de ce que l'on appelle leur « *classe* ». Par exemple on retrouve souvent dans les MMORPG, les classes « guerrier » (force physique et manipulation d'armes), « mage » (utilisation d'attaque magique à distance), « prêtre » (capacité à soigner) ou encore « archer » (attaque avec des armes à distance telles que les arcs).
- Enfin, pour le mode de jeu, nous avons choisi un jeu qui offrirait le mode multi-joueurs uniquement avec la spécificité de se limiter à deux joueurs ni plus ni moins. L'utilisation matérielle serait la suivante : chaque joueur possède sa machine et le jeu est joué en réseau local.

2. Étude des jeux similaires

Après cela, il nous a fallu avoir des idées d'inspiration plus précises. Nous avons alors effectué une étude complète des applications existantes similaires à notre futur produit. Quatre grandes catégories d'études se sont alors révélées nécessaires : l'étude de l'**originalité** et du **point fort du jeu**, l'**étude technique** des jeux du même type, l'**étude design** et **graphique** et enfin, l'**étude des choix** en terme **d'ergonomie**.

2.1. Étude des jeux similaires :

Nous avons essayé de détecter dans les jeux similaires que nous avons personnellement aimés jouer les points forts qui avaient fait de ces jeux des chefs-d'œuvre que nous n'oublierons pas. Nous pouvons par exemple citer :

2.1.1. Mario

« Mario » [voir figure 11], l'un des premiers jeux de plateformes .L'un des principaux points forts de Mario est son dynamisme. Les plateformes de sauts y sont parfois flottantes dans les airs et inaccessibles. C'est ce challenge qui a fait de Mario un grand titre. Nous avons également apprécié les ennemis de Mario [voir figure 12] qui sont très adaptés au monde des enfants du fait qu'ils sont comiques. De plus ayant choisi un univers forestier et fantastique, ce type d'ennemis nous



correspondait tout à fait.

« Figure 11 : Mario Brothers Wii »



« Figure 12 : un type d'ennemi du monde Mario »

2.1.2. Trine

Ensuite, le jeu « **Trine** » ([Frozenbyte, 2011](#)) [voir figure 13] possédant lui aussi les caractéristiques d'être en mode multi-joueurs, dans un univers fantastique et en 2.5D nous a paru un exemple pertinent. Trine est développé par un studio indépendant, « Frozenbyte », constitué de seulement 35 employés qui pourtant arrivent à rivaliser avec les grandes entreprises du domaine. La première particularité qui a fait de Trine un jeu extraordinaire est le jeu des lumières et des couleurs. En effet, le jeu, et particulièrement sa deuxième version, est très centré sur les « Halos lumineux » et les effets d'éblouissement de la caméra, ce qui crée une ambiance de magie et de rêve. Le deuxième point fort du jeu Trine et le jeu de la physique. Il est toujours surprenant dans un jeu de pouvoir interagir avec les autres objets du décor, les déplacer, les détruire en morceaux ou encore les pousser et le gameplay de Trine offre beaucoup de possibilités de ce type. Par exemple, le personnage du magicien de Trine peut, en traçant dans le ciel un cube, faire apparaître une caisse métallique dont il se sert pour transporter les autres personnages et joueurs.



« Figure 13 : Trine »

2.1.3. Giana Sisters

« **Giana Sisters** » ([Black Forest, 2012](#)) [voir figure 14] du studio indépendant « Black Forest » (le studio a choisi ce nom car le local est au beau milieu d'une forêt) est également un jeu de plateformes en 2.5D dans un univers fantastique. C'est un jeu graphiquement très beau qui propose une idée qui a beaucoup plu aux joueurs : le principe de la récupération de joyaux. Le personnage doit passer sur des joyaux disposés un peu partout sur la scène et parfois inaccessibles sans user de pouvoirs spéciaux et de l'habileté du joueur. Récupérer ces joyaux peut servir par exemple à ouvrir une porte fermée ou encore à augmenter le score du joueur.



« Figure 14 : Giana Sisters »

2.1.4. Zelda

Pour finir, « The Legend of Zelda » ([Hexa Drive, 2013](#)) est un autre jeu dont on a choisi de s'inspirer pour **The Twins**. Nous voulions qu'il y ait deux personnages dans notre jeu et « Zelda », possède justement deux personnages : Link le guerrier [voir figure 15] et Zelda [voir figure 16] la princesse dont l'apparence, très travaillée a plu à plusieurs générations d'enfants. Le jeu existe depuis l'époque des jeux 2D en « Pixel Art »²⁰, a été adapté par la suite sur toutes sorte de consoles telle que la « Nitendo DS »²¹ et existe aujourd'hui en version moderne du jeu sur la console « WII U » (les jeux de cette génération plus modernes sont communément appelés « current-gen games » ou « next-gen games »).



« Figure 15 : Link de la série Zelda »

« Figure 16 : Princesse Zelda de la série Zelda »

C'est de tous ces jeux que nous avons tirés les bases de l'imagination du jeu qu'est devenu **The Twins**. Nous avons premièrement essayé de combiner ces points forts des différents jeux d'une manière cohérente pour ensuite y ajouter une touche personnelle d'originalité.

2.2. Étude technique

Les jeux commercialisés dans le même type que le notre ne sont pas « open-source »²² et utilisent généralement des technologies propres au studio et non divulguées comme par exemple le UbiArt Framework²³. Il nous a donc été difficile d'étudier en profondeur le fonctionnement technique de ces jeux. Cependant, il existe deux moyens d'apprendre en partie le fonctionnement de ces applications :

²⁰ Objets composés à l'écran d'uniquement quelques pixels.

²¹ Console portable de Nintendo.

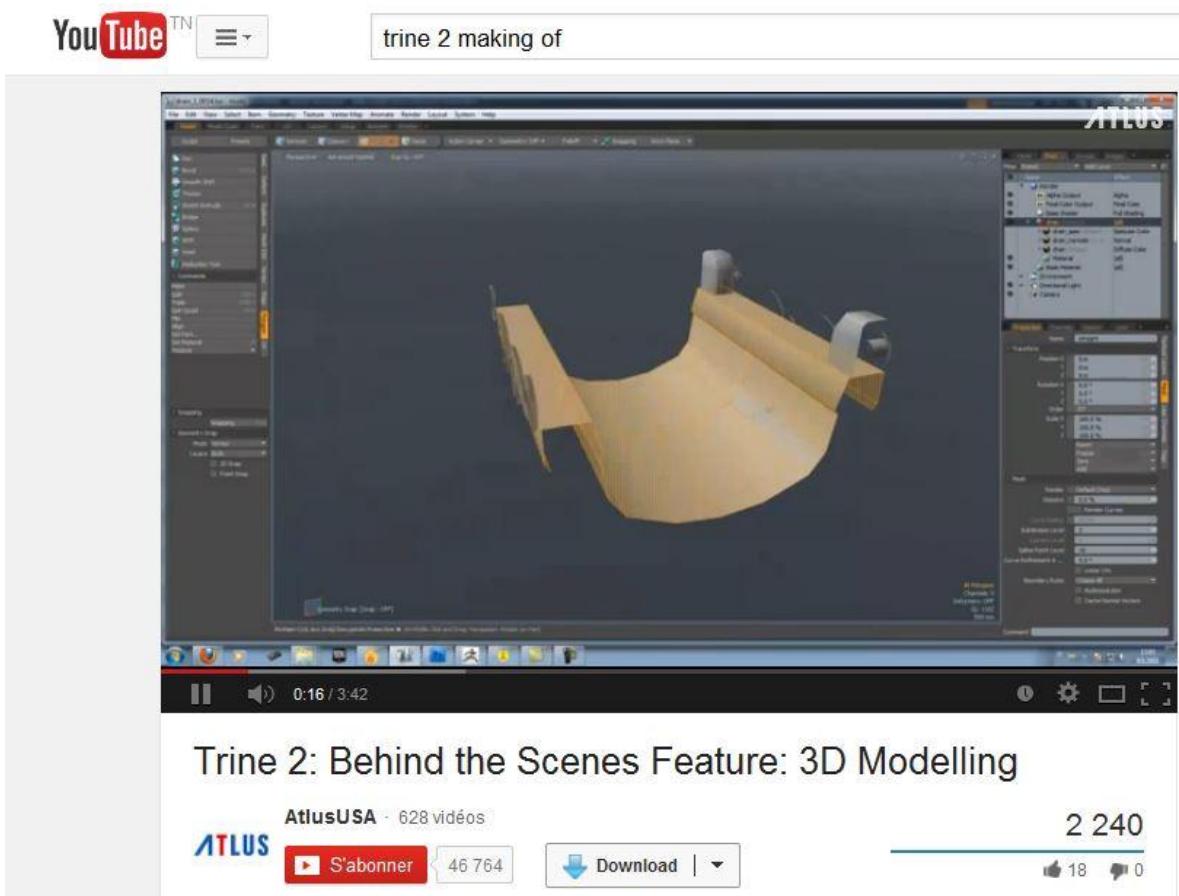
²² Code source du jeu accessible au public.

²³ Moteur spécialisé dans les jeux en 2D ou 2.5D du studio Ubisoft.

2.2.1. Les vidéos de Making Of : un moyen pour comprendre le fonctionnement des jeux

Les vidéos (sur Youtube ou autres) de type « **Making Of** »²⁴ [voir figure 17]. Ce sont des séries de vidéos dans lesquelles les développeurs d'un studio sont autorisés à dévoiler une partie de leur savoir-faire à tous ceux qui pourraient être intéressés. C'est grâce à ces chaînes Youtube que nous avons découvert notamment que :

- Dans les jeux de notre type, les scènes virtuelles dans lesquelles se déroule le jeu (appelé « **Map** »²⁵) sont en fait constituées de morceaux répétitifs recopiés plusieurs fois à l'identique et légèrement modifié par la suite pour éviter la sensation de répétitivité. On appelle cette technique le « **Modular Design** »²⁶.



« Figure 17 : vidéo making of modular design - Trine 2 »

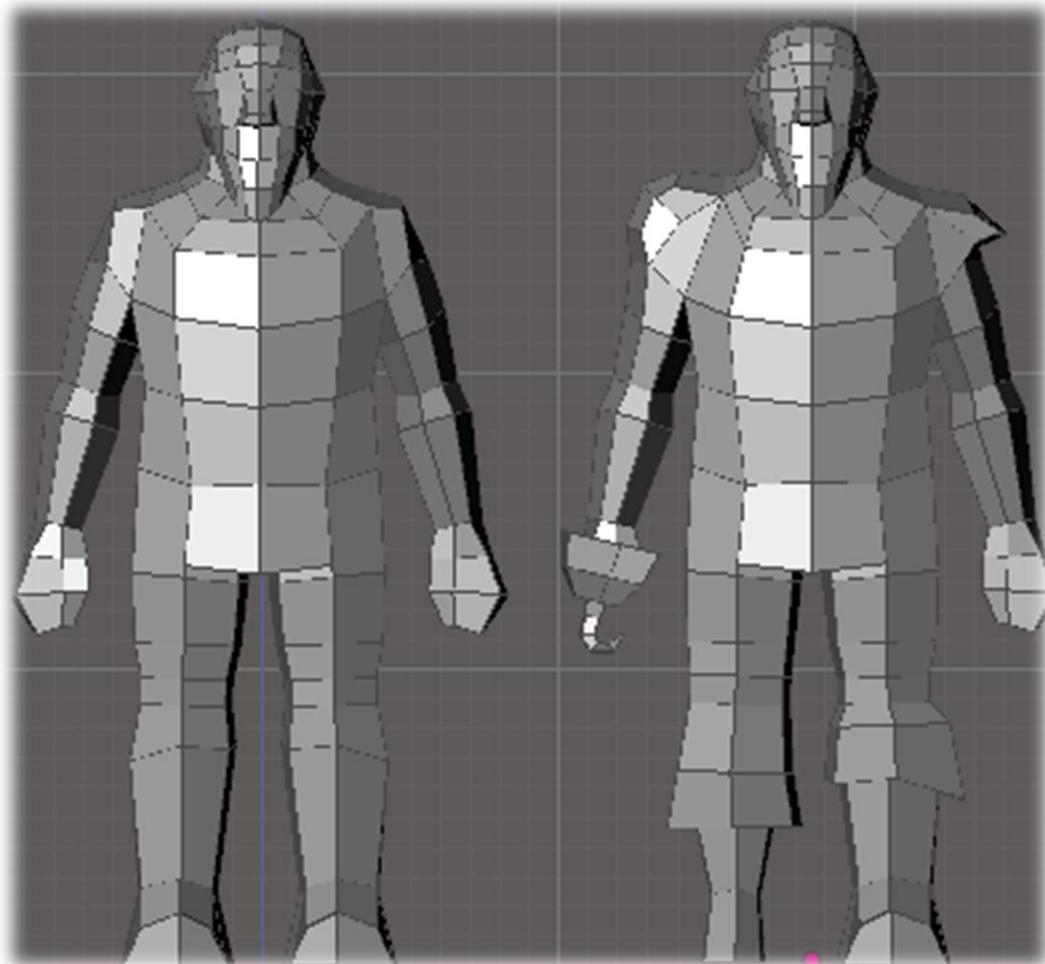
- Dans les jeux du même type que le notre, du fait les personnages ne se déplacent que sur deux axes, ils ne peuvent pas approcher les objets au fond de la scène et donc ne peuvent voir clairement comment ils sont faits. Ainsi, pour optimiser la mémoire et le temps de calcul du processeur, on ne crée pas réellement en 3D les objets au fond de la scène mais on les dessine sur les planes. Il faut savoir que plus un objet est complexe (nombre de polygones qui le compose), plus il prend de la mémoire et du temps à être calculé. Ainsi

²⁴ Vidéos montrant la réalisation technique de certains éléments d'une application multimédia.

²⁵ La carte du jeu.

²⁶ Design modifiable.

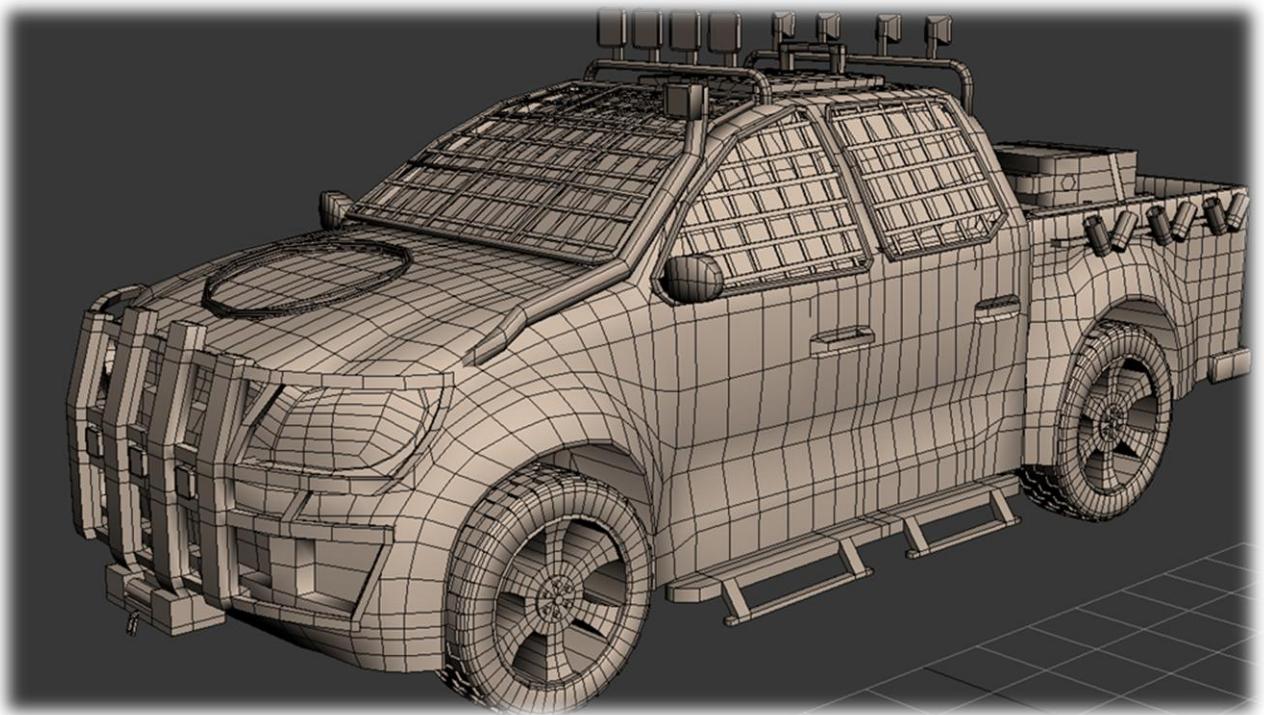
les objets 3D sont classés sous deux types « **low poly** »²⁷ [voir figure 18] (souvent mis en petite taille ou loin de la caméra pour que le joueur ne puisse en voir les défauts) et « **high poly** »²⁸ [voir figure 19] (souvent mis en avant car ils sont plus beaux).



« Figure 18 : modélisations low poly »

²⁷ Nombre de polygones réduit.

²⁸ Nombre de polygones élevé.



« Figure 19 : modélisation high poly »

- Dans les jeux de plateformes, les objets, étant petits, on utilise pour les **textures**²⁹ un format d'une taille multiple de huit comme par exemple la définition 512*512 pixels. Dans la 3D, il faut pour un seul objet une ou plusieurs textures : la « **Color map diffuse** »³⁰ pour la couleur, l'« **Alpha map** »³¹ pour définir les zones de transparence et accessoirement d'autres textures telles que la « **Specular Map** »³² qui définit les zones de l'objet qui reçoivent de la lumière et de l'ombre ou encore la « **Bump map** »³³ ou « **Normal map** » [voir figure 22] qui définissent les zones de l'objet qui doivent ressortir vers l'extérieur ou le contraire et ainsi ces textures créent un ensemble de petits détails tels que les imperfections de la peau humaine sans avoir à modéliser ces détails. Ces textures prennent la forme de l'objet 3D découpé et aplati. Dans le domaine des maquettes, on appelle cela des « patrons » [voir figure 20] et dans le domaine des objets 3D, on appelle cela des « UV » [voir figure 21]. Ce nom vient du fait que le modèle 3D créé sur trois axes (x, y et z) est découpé pour être coloré uniquement sur deux axes (u et v).

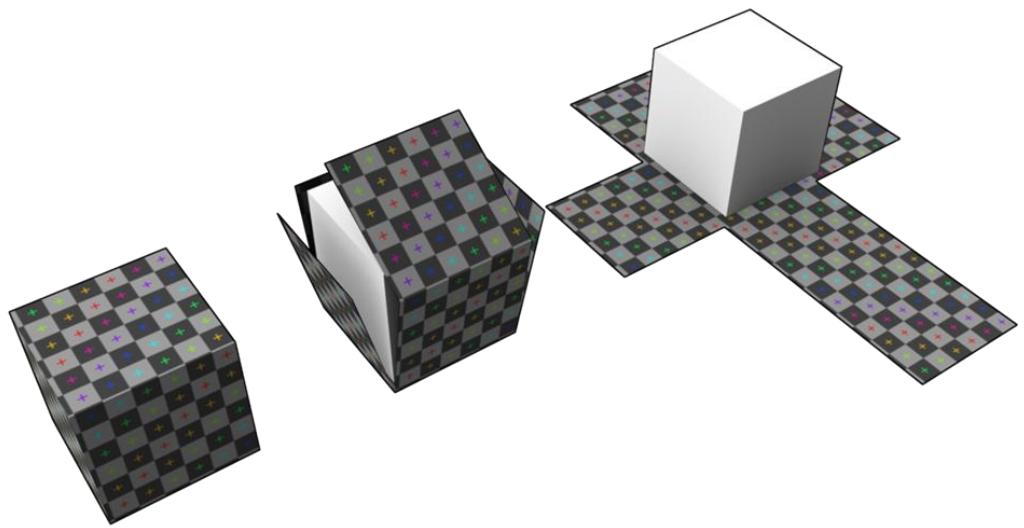
²⁹ Motifs et couleurs d'un objet 3D.

³⁰ Texture de diffusion de couleurs.

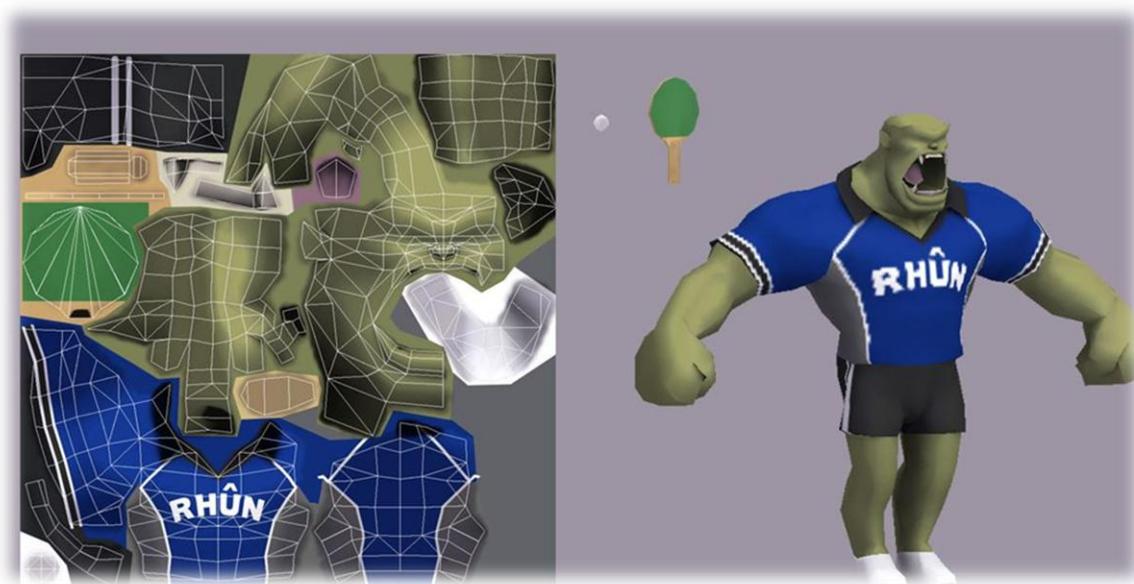
³¹ Texture gérant la chaîne alpha (transparence).

³² Texture définissant les surfaces (de l'objet 3D) qui reçoivent de la lumière.

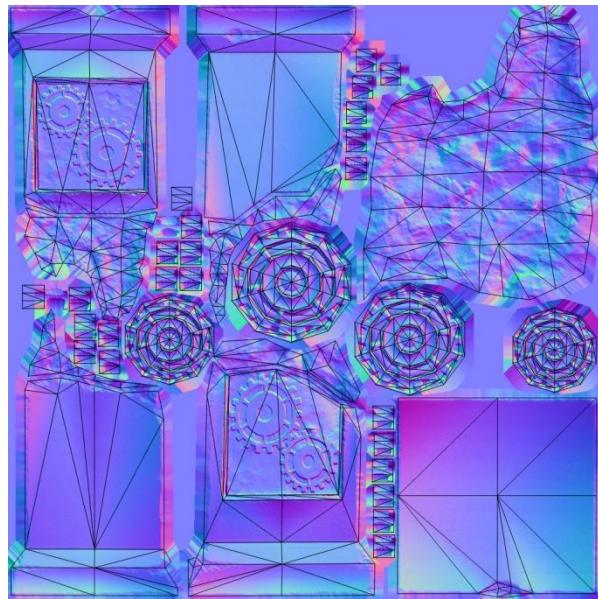
³³ Texture faisant ressortir certains détails sur la surface (de l'objet 3D).



« Figure 20 : Découpage des UV d'un cube »



« Figure 21 : UV et textures d'un personnage 3D »



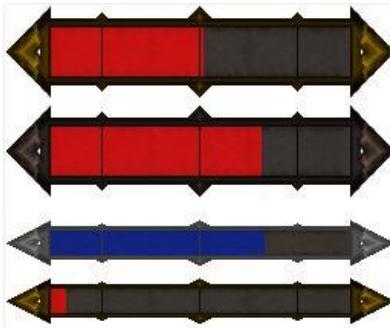
« Figure 22 : Normal map d'un objet 3D »

- Nous avons également appris que, malheureusement, certaines des technologies matérielles et logicielles utilisées ne serait pas à notre portée. Par exemple, au niveau matériel pour les animations, les studios professionnels utilisent des salles aux murs verts avec un grand nombre de caméras et de capteurs répartis sur les murs et un acrobate au milieu en costume spécial pour capturer et enregistrer ses mouvements. Au niveau logiciel, les studios développent eux-mêmes des logiciels adaptés à leurs besoins propres qu'ils ne partagent généralement pas.

2.2.2. Jouer : un moyen pour comprendre les techniques utilisées

À part les vidéos, tutoriels et making of, la deuxième méthode très pratique pour comprendre les techniques utilisées lors de la création est bien sûr de les essayer : de jouer. Ainsi, en testant des jeux du même type que le nôtre, nous avons remarqué beaucoup de détails techniques. Nous avons particulièrement noté les détails techniques liés au principe de jeu tels que par exemple :

- Dans tous les jeux de plateformes sur ordinateur, le saut se fait en utilisant la « barre espace », les déplacements en utilisant les flèches ou les touches « Z », « Q », « S » et « D » et les actions et attaques (attaquer, ouvrir une porte, activer un interrupteur...) se font en visant avec la souris et s'active en utilisant le bouton gauche de la souris. Les actions secondaires (comme la « protection ») se font en utilisant le bouton droit de la souris.
- Le niveau de santé d'un personnage qui détermine si le joueur a perdu ou si son personnage est toujours en vie se mesure soit à l'aide d'un nombre remarquable de « coeurs » soit à l'aide de ce que l'on appelle une **barre de santé** [voir figure 23].



« Figure 23 : barres de santé »

- Le menu « Pause » est remplacé dans les modes multi-joueurs par un menu « Options » qui ne met pas le jeu en pause mais qui affiche les fonctionnalités utilisées pour quitter la partie ou modifier les réglages visuels. Ce menu s'affiche toujours avec le bouton « Echap » du clavier.

2.3. Étude de l'ergonomie

Tout comme pour l'étude technique, nous avons étudié les différents jeux en les essayant. Il nous a vite paru évident que la facilité d'utilisation et le facteur « Agréable » d'un jeu devaient être séparés en deux études : les « **menus** » et les « **scènes de jeu** ».

2.3.1. Les menus

Pour les menus, l'objectif principal en termes d'ergonomie est que l'utilisateur arrive à distinguer sans avoir à chercher comment quitter, démarrer une partie ou encore mettre en pause.

- Ainsi on favorise le texte et pictogramme plutôt que les boutons de type iconographique. Les menus sont souvent affichés au centre de l'écran et y occupent la majorité de l'espace. [voir figure 24]
- De plus, si le fond des pages de menus est très coloré, se qui arrive quand les thèmes graphiques de menus sont en accord avec ceux du jeu, le texte des menus est de couleur blanche ou noire avec une grande taille de police pour faciliter la lecture.
- En outre, le sens de lecture y est souvent du haut vers le bas. Le bouton « Quitter » est souvent placé tout en bas de la liste. S'il n'est pas dans la liste, il se trouve en haut à droite comme celui d'une fenêtre « Windows ». Le bouton « Démarrer le jeu » est, au contraire, souvent placé tout en haut de la liste.
- De surcroît, si le fond de la page respecte le thème du jeu, on remarque que les écritures du menu sont généralement différencierées du reste par un fond noir peu opaque ou d'une couleur claire contrastant avec le reste de la page. [voir figure 25]



« Figure 24 : menu principale de Tom OConnor »



« Figure 25 : menu principal de Civilization 5 »

- Enfin, nous avons aussi remarqué un certain nombre de jeu dans lesquels les développeurs ont décidé de ne pas respecter ces règles afin de créer de l'originalité. Dans ces cas là, l'ergonomie est sacrifiée pour favoriser la beauté graphique de la scène et l'effet de surprise du joueur.

2.3.2. Les scènes de jeu

Pour les scènes de jeu, deux principaux critères d'ergonomie sont importants et forment à eux deux la facilité de compréhension de l'interface utilisateur présente dans le jeu.

- Le premier critère est la visibilité des possibilités d'actions et des personnages. Ce critère est d'autant plus important quand il s'agit d'un jeu pour enfant. Il faut par exemple que le personnage que l'on manipule soit remarquable par rapport aux autres et que les personnages contrôlables ou ennemis soient remarquables par rapport au décor. Pour ces raisons, le personnage que l'on contrôle se trouve généralement au centre de l'écran de jeu et peut être par exemple illuminé légèrement, plus net ou d'une couleur contrastant par rapport au décor. De plus

même les objets du décor doivent être différenciés visuellement en fonction de leurs utilités. Par exemple, pour que le joueur sache qu'un objet peut être détruit ou récupéré, l'apparence graphique du curseur change automatiquement au passage de la souris sur celui-ci. Dans les jeux où certains objets peuvent être récupérés, ils peuvent être accompagnés d'un « halo lumineux » et peuvent effectuer une légère animation (rotation continue) pour que le joueur les remarque [voir figure 26]. Enfin le chemin à suivre, doit être clair, sauf, dans le cas, évidemment, où le but du jeu est de trouver ce chemin (labyrinthe).



« Figure 26 : objet mis en valeur »

- Le deuxième critère et la rapide compréhension de ce que l'on appelle les « **GUI³⁴** » ou « **HUD³⁵** ». Les GUI présents sur la scène sont les indicateurs visuels utilisés par exemple pour la barre de vie, le nombre de joyaux récupéré ou encore le temps écoulé depuis le début de la partie. Ils sont souvent placés aux angles de l'écran en haut et en bas pour ne pas gêner la vue. Pour la même raison, les créateurs de jeux essaient de minimiser leurs tailles. Cependant, ils doivent être tout de même remarquables et leur intérêt doit être évident. Pour cela, on ajoute dans leurs formes propres ou à côté du GUI un symbole représentant l'objet qui lui est lié. [voir figure 27]



« Figure 27 : emplacements des GUI »

³⁴ Graphical User Interface : interface utilisateur graphique.

³⁵ Head-Up Display : affichage de premier plan.

2.4. Étude graphique

L'étude graphique des jeux vidéo est une tâche assez complexe dans le sens où il en existe un très grand nombre et tous différents sur le plan graphique. En effet, chaque studio tente de se faire une identité visuelle ou d'innover en créant un style nouveau. Dans l'optique de choisir l'exemple le plus proche de ce que nous cherchions, nous avons basé nos analyses graphiques sur le style pour enfants le plus abouti, celui qui berce l'enfance de la planète depuis si longtemps : le style **Walt Disney**.



« Figure 28 : Castle of Illusion »

Nous avons donc pour cela analysé le style de « **Castle of Illusion** » (Sega, 2013) [voir figure 28], jeu 2.5D pour enfants centré sur le personnage de « **Mickey Mouse** ». Les décors comme nous les recherchions sont forestiers et enchantés. Nous avons donc référencé les caractéristiques graphiques potentiellement intéressantes et essayé en même temps de comprendre les raisons de ces choix et leurs connotations :

2.4.1. La composition spatiale

Au niveau de la composition spatiale:

- Nous avons remarqué qu'il y a un personnage au centre de l'écran. Le personnage est vu de loin, la caméra fait donc un plan général et cela permet de voir au loin et d'anticiper les actions à venir.
- Le premier plan, devant le personnage est composé d'objets de hauteur basse pour ne pas encombrer la vue et cacher le personnage. L'utilité de mettre des objets au premier plan est de remplir le vide à l'écran et de camoufler la coupure due au fait que le jeu est en 2.5D. En effet, plus le premier plan est rempli, plus le joueur à l'impression d'être à l'intérieur du monde du jeu.
- Le second plan est presque vide mais plus éclairé que le reste de la scène. Ainsi, le joueur devine où il doit aller car le chemin est mis en avant et globalement dégagé.
- Le troisième plan et le fond de l'image sont assombries par rapport au chemin, ils sont remplis d'objets plus grands (des arbres et des murs) qui forment l'ambiance principale de la scène

forestière et qui, en même temps, sont utilisés pour cacher la ligne droite d'horizon. Le fond possède également un effet de brouillard qui, comme les arbres, ajoute un supplément à l'ambiance et cache la coupure de la ligne d'horizon.

2.4.2. Les formes et les objets

Au niveau des formes et des objets :

- Les différents types d'objets présents sur la scène sont : végétation (bois et feuilles), des objets manufacturés en bois (les grilles de la porte), des pierres naturelles et des constructions en pierres. On remarque une certaine homogénéité et cohérence entre les types des objets cités. Cela donne une touche de réalisme qui aide l'enfant à s'imprégner facilement du monde du jeu.
- Le personnage au centre de l'écran est une icône du monde Disney : « Mickey Mouse ». Le personnage est une personnification d'un animal, ce qui n'est d'ailleurs pas en contradiction avec l'univers forestier. Il a les formes rondes et très peu de détails remarquables mis à part ces oreilles et cela est également valable pour ses vêtements. Son visage et ses expressions sont très enfantins et naïfs. Il a donc été créé pour plaire à un maximum de personnes et surtout aux enfants. Les formes de la tête du personnage se retrouvent sur les poignées des portes et se retrouvent sûrement à beaucoup d'autres endroits dans le jeu. Ce sont des rappels éparpillés un peu partout dans le monde du jeu pour insister sur l'appartenance de ce dernier à Disney.
- Les formes, de manière générale, sont simples et les créateurs en ont profité pour créer des formes agréables et attachantes pour les enfants.

2.4.3. Couleurs

Au niveau des couleurs utilisées:

- Les couleurs des objets du premier et troisième plan sont le gris, le marron et le vert. Ce sont des couleurs basées sur ce qu'elles sont réellement dans la nature mais avec un effet de peinture pour qu'elles ne soient pas trop proches du monde réel et pour ressembler au monde des dessins animés Disney. De plus, ces couleurs sont soigneusement choisies pour être « confortables » dans le sens où il y a d'une part un fort contraste entre les valeurs (les gris) et les tons (les verts) et d'autre part, une homogénéité entre les différents tons.
- Les couleurs du fond sont les mêmes mais il y a en plus un brouillard de couleur bleu. Le bleu est choisi car il est la couleur du ciel qui n'est pas vraiment visible dans cette scène et car peu présent dans la nature, il donne une impression de magie et de mystère.
- Les couleurs du personnage sont très vives (le jaune et le rouge) et sont encore plus remarquables avec la présence du blanc et du noir. Le personnage est ainsi bien visible et se démarque du décor. Le jaune et le rouge ont été également choisis en fonction du public enfantin car ce sont des couleurs très attirantes (utilisées notamment pour le maquillage du clown, l'un des personnages clé pour les enfants).
- Enfin, on remarque qu'il y a une source lumineuse à la gauche de la scène, à l'intérieur de l'enceinte des murs de pierres. Sachant que dans ce type de jeu, on se déplace généralement de la gauche vers la droite, on peut déduire que « Mickey » vient de cette zone. Cette lumière crée donc chez le joueur la sensation d'avoir quitté une zone sûre et d'être entré dans un monde sombre et donc mystérieux ou plein d'aventures.

Toutes ces études nous ont permis d'accéder à une liste de règles graphiques, techniques ou ergonomiques nécessaires pour obtenir de notre travail un résultat suffisamment professionnel. Ainsi nous en avons tenu compte dans les phases suivantes de spécification et de conception.

3. Les sites web, un outil de distribution

Plusieurs chemins existent pour amener le jeu depuis le studio de production jusqu'aux joueurs. Il existe, par exemple, des logiciels tels que les plates-formes de distribution de jeux. « **STEAM** » développé par le studio de production de jeu « **Valve** » est l'une des ces plates-formes spécialisées dans les jeux pour PC et les autres studios ayant développé un opus peuvent ou non décider de le distribuer dessus. Il existe ensuite la méthode très courante de distribuer le jeu sur « **Blu-ray** »³⁶ dans les magasins spécialisés dans le jeu (par exemple « **Micromania** »³⁷). Cependant, les supports du web restent tout de même les plus accessibles pour les développeurs. En effet, aucun fond n'est nécessaire, si ce n'est l'hébergement, pour un petit studio qui souhaiterait être présent sur les réseaux sociaux ou avoir son propre site web. Ainsi, grandes ou petites, toutes les entreprises du domaine possèdent une page **Facebook** et **Twitter** pour leurs entreprises et parfois une page même pour chacun des jeux qu'elles créent. Par ailleurs, les studios possèdent généralement une chaîne **Youtube** pour présenter leurs jeux. On trouve généralement dans ces pages des images des jeux en développement, des articles sur les jeux ou sur l'entreprise, des tests et évaluations des jeux, des bandes annonces, des « **making of** », des demandes d'avis aux fans de la page [voir figure 29] ou encore des annonces et messages du **service commercial** adressés aux clients pour créer un dialogue, un lien de proximité et donc une fidélité.



« Figure 29 : un post sur la page communautaire de Watch_Dogs »

³⁶ Support d'une capacité de stockage très élevée.

³⁷ Magasin de vente de jeux vidéo.

Pour ce qui est des sites web pour jeux, deux solutions s'offrent au studio :

3.1. Une sous-rubrique dédiée

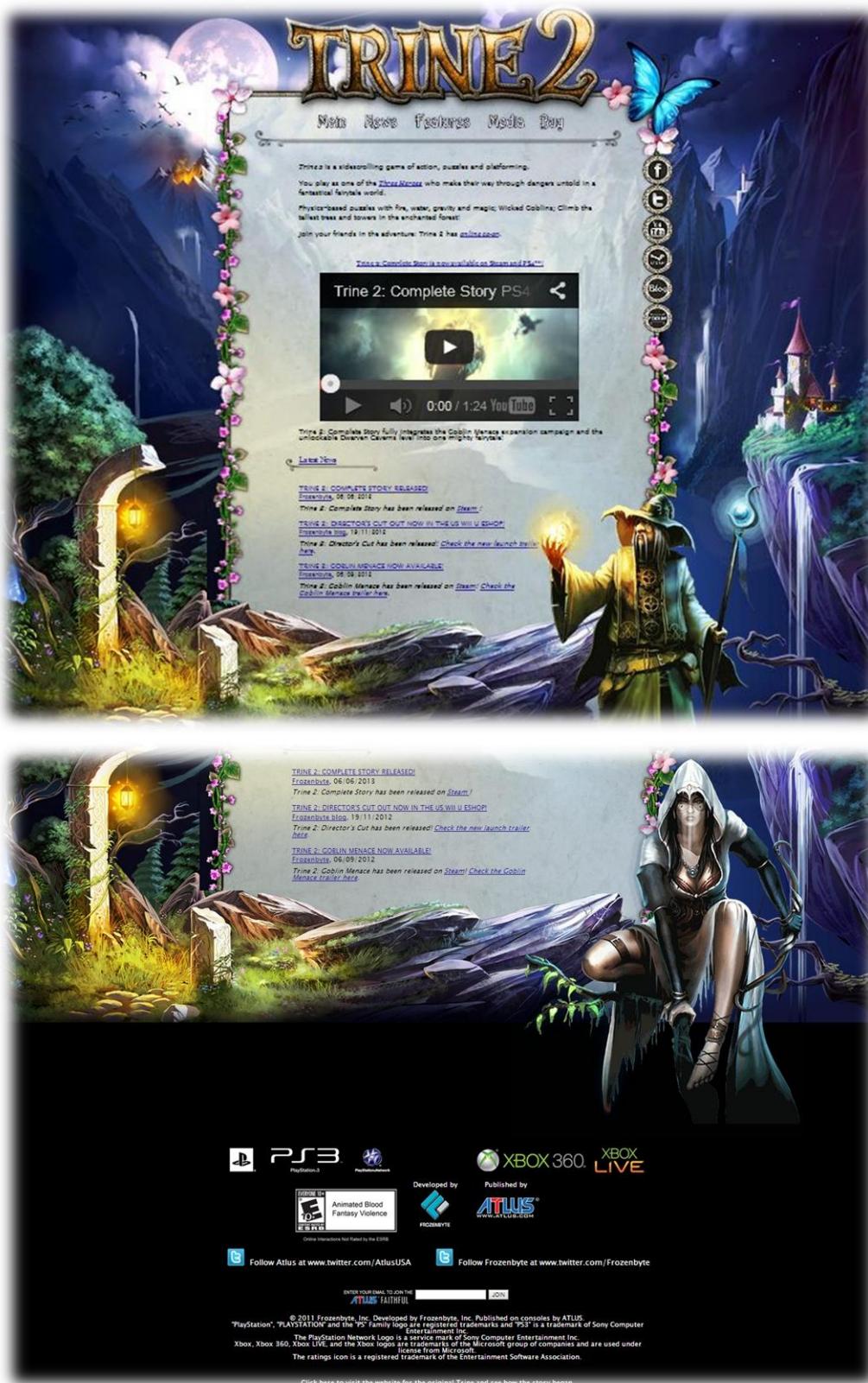
Ils peuvent créer une sous-rubrique pour chacun des jeux qu'ils ont développés. Dans ce cas, cette rubrique étant liée au reste du site web, elle doit être en accord à la fois avec le thème général du site et avec le jeu présenté. Ainsi, les créateurs de ces sites web tiennent compte de ce fait et essaient de choisir des couleurs qui ne généreraient pas quelques soit les couleurs du jeu présenté (les niveaux de gris, le noir et le blanc correspondent tout à fait). Ils favorisent aussi les formes simples comme les formes carrées pour qu'elles ne changent rien à l'ambiance quelque soit le jeu. [voir figure 30]



« Figure 30 : page d'accueil du site web de Frozenbyte »

3.2. Un site web dédié

L'autre solution est de créer tout un site web dédié à un seul jeu ou à une saga (suite d'épisode d'un même titre). Dans ce second cas, les sites possèdent un design entièrement dédié au jeu et donc plus fantasiste [voir figure 31]. Cette technique permet de limiter l'aspect commercial (car le studio n'est pas mis en avant) et favorise la création d'une ambiance qui provoquera l'envie de jouer. Cette méthode permet aussi du côté des développeurs de faciliter la création car le site peut ainsi être statique (avec une simple intégration d'un kit de paiement pour ceux qui voudraient acheter). En effet, contrairement au site de l'entreprise qui doit évoluer à la sortie de chaque nouveau produit, le site web d'un jeu reste souvent le même au fil du temps.



« Figure 31 : aperçu du site web de Trine 2 »

- Cependant, il est tout à fait possible qu'une société décide de combiner les deux sites web pour maximiser la diffusion de leurs produits.

Enfin, il est primordial pour l'entreprise de lier tous ces supports ensemble et de créer un réseau. Il est par exemple possible d'ajouter des « liens hypertextes » vers la page Facebook dans le site web et de partager les actualités du site web dans la page Facebook.

3.3. Bilan et choix

Après ces recherches et parmi toutes les possibilités présentées précédemment, nous avons jugé qu'il serait adéquat pour nous de créer :

- Pour notre groupe de travail renommé « Prod'IT Studio » : une page « Facebook » et « Google+ » pour communiquer les nouveautés avec les fans comme une véritable entreprise, une adresse « Gmail » et une chaîne Youtube pour héberger les vidéos annonçant nos jeux à commencer par « **The Twins** ».
- Pour notre jeu « **The Twins** » : un site web dédié avec le thème du jeu.

Ces choix nous ont conduits à étudier plus en détails les sites web dédiés aux jeux pour avoir une idée plus précise de ce qu'ils doivent contenir et de la manière dont ils sont réalisés. Dans cette optique, nous avons étudié le site web du jeu « Trine 2 », illustré ci dessus. [voir figure 31]

4. Étude des sites web similaires

4.1. Analyse technique :

Le site web de « Tine 2 » possède un schéma de navigation en **réseau (non hiérarchisé)** assez simple, il y a cinq pages et de chacune d'entre elles, on peut aller vers les autres.

Ces cinq pages sont : « Main » page présentant le jeu de manière générale, « News » dans laquelle est affichée une liste des évolutions du jeu classées par dates, « Features », page dans laquelle on trouve les fonctionnalités que le jeu permet, « Media », page contenant des images et vidéos du jeu et enfin, « Buy », page dans laquelle on peut télécharger le jeu dans ses différentes versions (en payant s'il n'est pas gratuit).

Le site est développé en **PHP** mais seule la page « News » est réellement susceptible d'évoluer, le site est donc techniquement assez simple. Les images de la page « Media » ainsi que les exécutables à télécharger de la page « Buy » sont enregistrés dans la même machine serveur que le site mais les vidéos, quant à elles, ont été enregistrées sur **Youtube** puis un lien généré par **Youtube** lui-même est mis dans la page.

4.2. Étude graphique et ergonomique

- Nous ne parlerons que très peu des couleurs du site web car leurs choix et leurs significations sont propres au jeu et n'apportent donc pas un savoir ou une règle à respecter. Nous dirons simplement que pour faciliter la lecture, tout comme les menus des jeux, le texte est écrit sur un fond qui permet la lisibilité et possède une couleur (ici le noir) remarquable par rapport au reste de la page.
- Au niveau de la composition spatiale, le site n'est composé que d'un sens d'écriture. Le site se lit de **haut en bas**. On commence par le nom du jeu en très grand, puis les menus, un peu

moins grands mais toujours remarquables par rapport au texte simple et on finit par le texte contenu dans la page. En bas de chaque page, se trouve une liste de logos sur fond noir. Le fond est noir pour ne pas gâcher les couleurs diverses et variées de ces différents logos et parce que ces derniers n'appartiennent pas directement au jeu. Ce sont en fait les logos des différents associés et partenaires du studio et en même temps un lien vers leurs propres pages web. [voir figure 32]

- Le fond de la page, bien qu'il possède les couleurs du jeu, est une **peinture digitale** et non pas une prise des graphismes 3D du jeu. Cela donne le sentiment d'être entré dans l'intimité de l'équipe de développement et d'avoir accès à leurs travaux. Aucune symétrie parfaite n'est faite : on remarque une liste de liens externes est visible sur la droite mais pas sur la gauche, un personnage se trouve en bas à droite mais il n'y a rien de l'autre côté et enfin, par endroit, le fond se mélange avec le plan de lecture et passe même devant. Cela donne une impression de tri-dimension se qui est en accord avec le jeu.
- Cependant tout comme pour les couleurs, les choix des formes et des objets ne nous concernent pas vraiment ici car ils sont eux aussi en liaison avec le jeu présenté et n'apportent donc pas un chemin à suivre.

Avoir étudié le site web de Trine 2 nous a donc permis d'accéder à des indications sur les différentes pages que nous devions trouver sur notre site web, sur le type de schéma de navigation propice pour ce type de site ainsi que pour une idée de composition spatiale adéquate pour notre site web.

Ces études terminées, nous avions une idée globale du milieu dans lequel nous comptions travailler, le monde du jeu, sur les différentes méthodes de travail à suivre ainsi que sur ce que serait précisément le produit que nous allions créer. Nous pouvions donc commencer les phases de spécification de notre application.



The Twins

Dans cette deuxième partie, ayant collecté suffisamment de connaissances, nous étions prêts à concevoir notre application. Le choix d'utiliser « scrum » comme processus de travail, un cycle de vie agile incrémental, nous a conduits à séparer l'analyse et la conception de l'application dans les différents « sprints ». Ainsi, chaque incrément doit être un exécutable autonome possédant sa propre phase d'analyse et de conception. Par contre, nous souhaitions, avant de commencer cette aventure que fut **The Twins**, effectuer tout de même une conception générale. Et dans cette partie, nous vous présenterons de manière groupée les spécifications du jeu ainsi que sa conception.

Cette conception globale de **The Twins** nous permettait d'accéder à plusieurs avantages qui nous paraissaient primordiaux :

- Avoir un synopsis et une idée suffisamment détaillée pour la faire valider par nos différents Encadrants : Madame Jihene Malek de L'ISAMM ainsi que l'équipe de StolenPad Studio.
- Effectuer le Backlog du produit d'une manière réaliste et donc avoir une idée sur l'ensemble des tâches que nous devrions exécuter. Après quoi, il nous était plus facile d'établir un planning pour les différents sprints et nous répartir équitablement les tâches.
- Nous décider sur les moyens qui devraient être mis en œuvre par la suite et ainsi avoir d'avance ces outils à notre disposition.

1. Synopsis de l'application « The Twins »

Nous avons alors commencé par faire un synopsis classique en faisant abstraction de la spécificité du produit ce qui permet d'apporter des réponses aux questions auxquelles on aurait pu ne pas penser en gardant à l'esprit la nature d'un jeu.

1.1. Axe de communication

- But du produit (objectifs généraux): le but d'un jeu vidéo est la détente de l'utilisateur en lui apportant une expérience à la fois émotionnelle, intellectuelle et dans notre cas sociale. (www.psyetgeek.com)
- Objectifs personnels (en temps qu'étudiants mais aussi en temps que Prod'IT Studio) : réussir à créer un produit de qualité professionnelle et à le distribuer par la suite.
- Clientèle cible : enfants âgés entre neuf et quatorze ans des deux sexes sans distinction. Nous n'avons pas établi une région ciblée car le jeu est distribué via Internet, cependant n'existant qu'en version française, le client sera de préférence francophone. Le jeu étant gratuit, de type «Free To Play », nous ne faisons pas non plus de distinction sociale.
- Contexte d'utilisation : les jeux vidéo pour enfants sont généralement jouer à la maison (lieu de résidence) du client. Notre jeu se jouera sur ordinateur personnel pour support matériel, celui de l'enfant ou s'il n'en possède pas à cause de son jeune âge, sur celui de ses parents.

1.2. Sujet

- But du jeu : traverser la forêt et arriver dans un village le plus vite possible et faire le meilleur temps.
- Personnages : les personnages sont « Umi » (en japonais l'« océan ») et « Almas » (en arabe « les joyaux ») des frères et sœurs jumeaux qui habitent dans un village au milieu de la forêt. Umi, la sœur possède d'utiles pouvoirs magiques qui lui permettent de franchir les pièges de

la forêt tandis qu'Almas possède différentes armes dont il se sert pour protéger sa sœur des plantes carnivores.

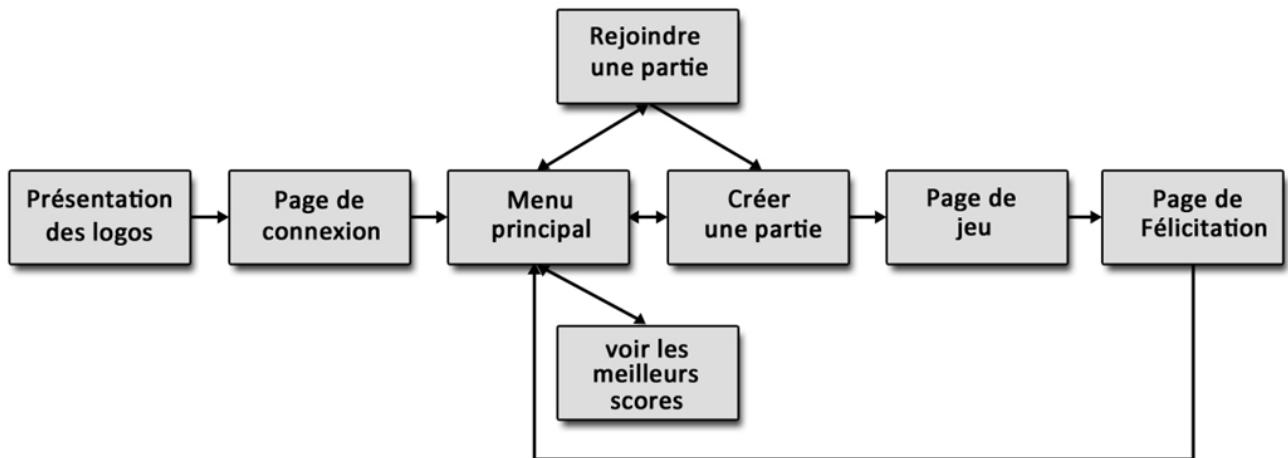
- Adversaires : les adversaires sont des plantes carnivores dont certaines ont la capacité de voler grâce à leur feuillage. Ces plantes essaient d'empoisonner Umi et Almas.
- Lieu : le jeu se déroule dans une forêt peuplée de papillons scintillants. À la fois belle est dangereuse, on y trouve des lacs empoisonnés, des pièges et diverses sortes de plantes.
- Scénario : voici le scénario tel qu'il est dans le jeu et le manuel [voir figure 32] :

« Depuis plusieurs jours déjà, le village à l'ouest de la mystérieuse forêt a décidé de fermer ses portes. Les temps sont rudes, particulièrement depuis que les plantes carnivores sont sorties des eaux pour envahir la forêt. Elles sont de plus en plus nombreuses et se rapprochent de jours en jours. La situation est critique, elles ont passé la veille les deux ponts et seront bientôt arrivées aux portes de la ville. Les habitants ont décidé d'envoyer Almas et Umi, les Twins pour demander de l'aide à leurs voisins vivants à l'est de la forêt. Les Twins devront faire vite car le danger grandit à chaque seconde. Malheureusement pour eux, la forêt est vieille et mystérieuse. On raconte qu'elle serait parsemée de pièges et d'éénigmes. Seule leur inébranlable fraternité et une entraide sans faille leur permettra de les franchir et de mener à bien leur mission. »

« Figure 32 : scénario »

1.3. Description visuelle et sonore

- Plan de la caméra : la caméra fait un plan générale, elle vise en son centre le personnage joué qui est vue de profil. La caméra ne change jamais d'angle, elle se trouve légèrement plus haute que le personnage et n'est donc pas tout à fait horizontale.
- Sons du jeu : n'ayant pas reçu des cours en matière de son, nous avons préféré simplifier la partie sonore. Nous avons donc choisi de composer une musique d'ambiance inspirant le rêve et le mystère qui se jouera en boucle lors des parties de jeu.
- Schéma de navigation de l'application : parmi tous les schémas de navigation qui s'offraient à nous, celui qui s'orientait le plus facilement vers les enfants est le schéma « linéaire » (une structure séquentielle). En effet cette structure permet en quelques sortes de guider l'utilisateur qui, ainsi, ne risque pas de se perdre. Cependant, suite à la présence d'un menu, étant obligatoire dans notre application, nous avons du lier la structure linéaire à celle d'un menu. Dans cette même optique de simplifier le choix à l'enfant et de minimiser les possibilités de perte, nous avons choisi un menu de type « centralisé » appelé aussi « en étoile » sans aucun sous-menu. Le voici tel que nous l'avions imaginé [voir figure 33] :

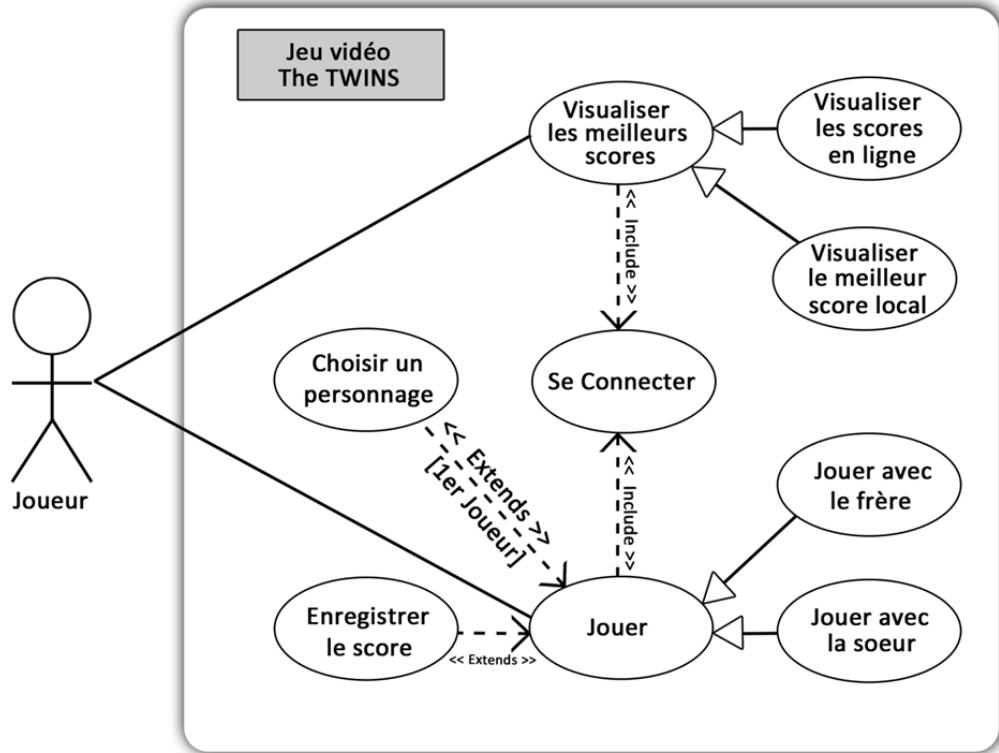


« Figure 33 : schéma de navigation de The Twins »

Après avoir complété ce synopsis, nous passâmes à l'étape technique de la phase de spécification : l'utilisation du langage UML. Le fait que notre jeu devait comporter une base de données et serait développé selon les règles de la programmation orientée objets, a fait du langage UML le choix le mieux adapté.

2. Spécifications fonctionnelles

Nous commençâmes dans l'ordre logique de la tâche et avons donc établie le **diagramme des cas d'utilisation** (« **Use Case** ») [voir figure 34] du jeu **The Twins**. Ensuite, nous avons détaillé textuellement les fonctionnalités les plus importantes, possédant plusieurs déroulements possibles ou ayant besoin d'un éclaircissement.



« Figure 34 : diagramme de cas d'utilisation »

2.1. Explication complémentaire

Dans le cas où les noms des cas d'utilisation ne seraient pas suffisamment explicatifs, nous allons ajouter une brève explication quant à la manière dont nous avons imaginé ces fonctionnalités :

- Pour garantir que le joueur a bien téléchargé le jeu à partir de notre site web et qu'il s'est inscrit dans notre base de données, on demande au joueur une connexion sur son compte Twins. Ainsi, si nous souhaitons rendre le jeu payant, le paiement se fera lors de la création du compte sur le site web après quoi le joueur démarrera le jeu en se connectant.
- Deux types d'enregistrement de score sont possibles : les scores que le joueur fait sont enregistrés automatiquement sur son ordinateur mais il peut également choisir de les enregistrer en ligne pour les rendre visibles à tous les joueurs ou les visiteurs du site web du jeu.
- Le jeu se joue à deux, le premier joueur choisit son personnage (« Umi » ou « Almas ») et le deuxième, celui qui rejoint la partie, hérite donc automatiquement du personnage inutilisé. Les possibilités d'action divergent entre les deux personnages.

Dans l'intérêt d'expliquer la différence de jeu entre les deux personnages ainsi que le processus de connexion, nous allons ajouter à ce diagramme les descriptions textuelles de ses principaux cas d'utilisation.

2.2. Descriptions textuelles

2.1.2. Description textuelle du processus de connexion

Sommaire d'identification

Titre : Connexion.

But : utiliser le jeu The Twins.

Résumé : un joueur s'identifie via son pseudo et son mot de passe.

Acteurs : un « Joueur ».

Date de mise à jour : 8 février 2014.

Responsables : Anas Neumann et Achref Bouhadida.

Description des enchaînements

Pré-conditions : être connecté à Internet, avoir démarré le jeu et possédé un compte créé sur le site web du jeu The Twins.

Enchaînement nominal :

- 1) Le joueur entre son pseudo et son mot de passe dans leurs champs réservés respectivement,
- 2) Le jeu effectue une vérification,
- 3) Le joueur est redirigé vers le menu principal.

Enchaînement alternatif n°1 : erreur dans le couple pseudo / mot de passe.

- 3) Le système a détecté une erreur dans le couple pseudo / mot de passe,
- 4) Un message d'erreur est affiché à l'écran,
- 5) Le joueur doit réessayer et revient donc à l'étape 1).

Post-conditions : le joueur est connecté.

Description des besoins IHM

Besoin IHM : champ de saisie, bouton de validation et message d'erreur visible et compréhensible.

Besoin non-fonctionnels : connexion en ligne suffisamment rapide.

« Figure 35 : description textuelle du processus de connexion »

2.1.3. Description textuelle du jeu avec le personnage «Almas», le frère :

Sommaire d'identification

Titre : Jouer en tant que « frère ».

But : finir la partie pour effectuer un score.

Résumé : un joueur joue avec le personnage « Frère » avec un autre joueur qui joue avec le personnage « Soeur »

Acteurs : un « Joueur ».

Date de mise à jour : 8 février 2014.

Responsables : Anas Neumann et Achref Bouhadida.

Description des enchaînements

Pré-conditions : un joueur ne peut faire un score à lui tout seul. Il faut, pour jouer, être deux joueurs reliés par un réseau local. Pour jouer en tant que frère, il faut avoir soit créé soit rejoint une partie.

Enchaînement nominal :

- 1) Le personnage « Almas » est instancié sur la scène de jeu.
- 2) Il déplace le personnage vers le village à droite avec la touche « D » ou la flèche droite.
- 3) Quand le personnage se trouve face à un obstacle ou un vide, le joueur appuie sur la « Barre espace » pour faire sauter le personnage.
- 4) Quand le personnage se trouvera devant une vigne dégageant des vapeurs de poison, il attend que le joueur « Umi » ouvre le passage.
- 5) Quand le personnage se trouve face à une plante carnivore, il se sert de la souris pour attaquer la plante et la détruire.
- 6) Une fois arrivé au village, la partie est finie et le joueur a joué une partie en tant que « frère ».

Enchaînement alternatif n°1 : zone inaccessible

- 4) Après avoir essayé de franchir l'obstacle, le joueur se rend compte qu'il ne peut atteindre sa destination.
- 5) Le joueur est bloqué tant que le joueur « Sœur » n'a pas créé un cube magique lui permettant de franchir la zone.
- 6) Une fois ceci fait, le joueur fait monter son personnage sur le cube et traverse la zone.
- 7) Le scénario nominal reprend à l'étape 5.

Enchaînement alternatif n°2 : Mort du personnage

- 5) Une plante a enlevé toute la vie du garçon, il meurt et revient tout au début.
- 6) Le scénario nominal reprend donc à l'étape 2.

Post-conditions : La connexion du joueur ne doit pas être interrompue durant tout le long de la partie.

Description des besoins IHM

Besoin IHM : beauté de la forêt, multitude de couleurs.

Besoin non-fonctionnels : un taux d'au moins 40 envois par seconde pour éviter de gêner les joueurs avec un retard réseau (latence).

« Figure 36 : description textuelle du jeu avec le personnage « Almas » »

2.1.4. Description textuelle de jeu en temps que « Umi », le personnage sœur :

Sommaire d'identification

Titre : Jouer en temps que « Sœur »

But : finir la partie pour effectuer un score.

Résumé : un joueur joue avec le personnage « Sœur » avec un autre joueur qui joue avec le personnage « Frère ».

Acteurs : un « Joueur».

Date de mise à jour : 8 février 2014.

Responsables : Anas Neumann et Achref Bouhadida.

Description des enchaînements

Pré-conditions : un joueur ne peut faire un score à lui tout seul, il faut pour jouer être deux joueurs reliés par un réseau local. Pour jouer en temps que « Sœur », il faut avoir soit créé soit rejoint une partie.

Enchaînement nominal :

- 1) Le personnage « Umi » apparaît sur la scène et le joueur peut alors le contrôler.
- 2) Quand une plante approche de « Umi », le joueur peut la glacer à l'aide du clic gauche de la souris.
- 3) Si le personnage « Almas » est bloqué, le joueur peut tracer un cube dans les airs afin de l'instancier dans le jeu.
- 4) La fille peut ensuite déplacer le cube, et le garçon s'il se trouve au dessus, à l'aide du clic gauche de la souris.
- 5) Si la fille est bloquée par un obstacle, elle peut passer à travers et arriver de l'autre côté avec le clic droit de la souris, c'est la « Téléportation ».
- 6) Une fois arrivé au village, la partie est finie et le joueur a joué une partie en tant que « Sœur ».

Enchaînement alternatif n°1 : échec du cube.

- 2) Le tracé du cube est faux ou trop lent, le cube n'apparaît pas et le joueur doit recommencer.
- 3) Le scénario nominal reprend à l'étape 4.

Enchaînement alternatif n°2 : mort du personnage.

- 2) Une plante a réussi à enlever toute la vie de « Umi »,
- 3) Le personnage meurt et revient tout au début du jeu,
- 4) Le scénario reprend donc à l'étape 2.

Post-conditions : la connexion du joueur ne doit pas être interrompue durant tout le long de la partie.

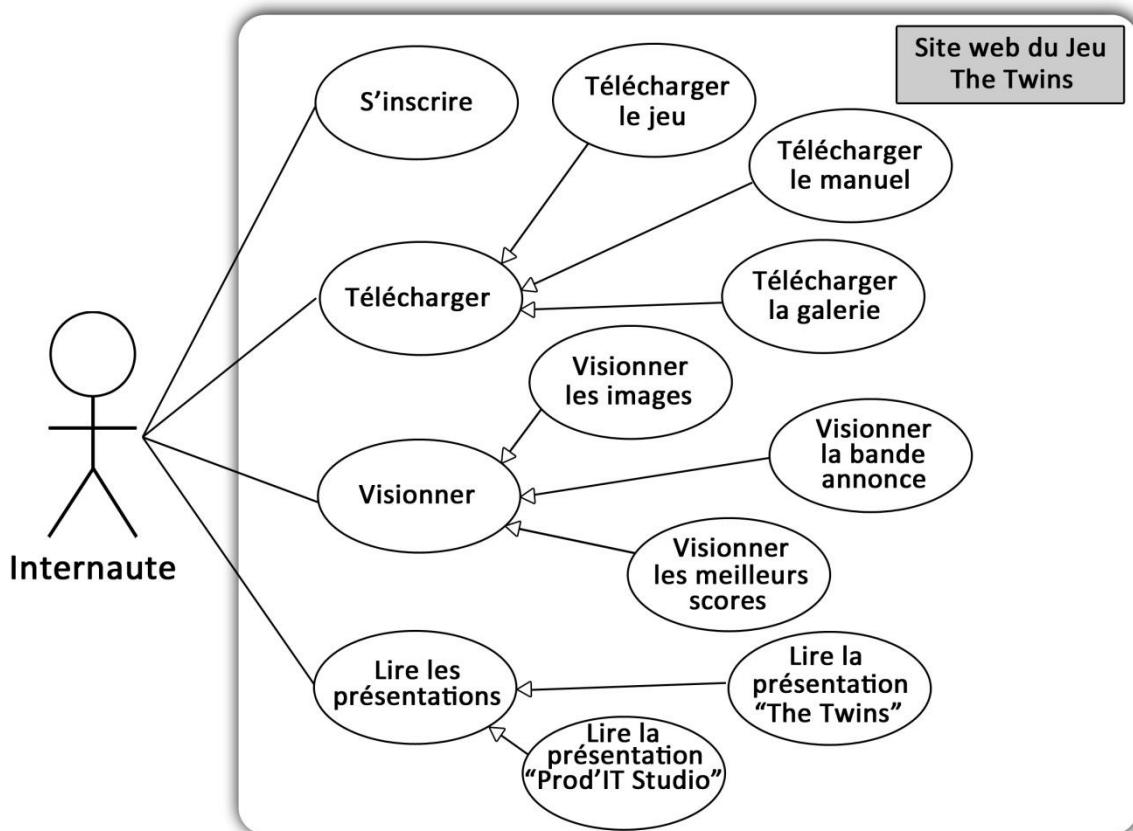
Description des besoins IHM

Besoin IHM : beauté de la forêt, multitude de couleurs.

Besoin non-fonctionnels : un taux d'au moins 40 envois par seconde pour éviter de gêner les joueurs avec un retard réseau (latence).

« Figure 37 : description textuelle du jeu avec le personnage « Umi » »

2.3. Diagramme de cas d'utilisation du site web



« Figure 38 : diagramme de cas d'utilisation »

Le site web a cinq principales utilités :

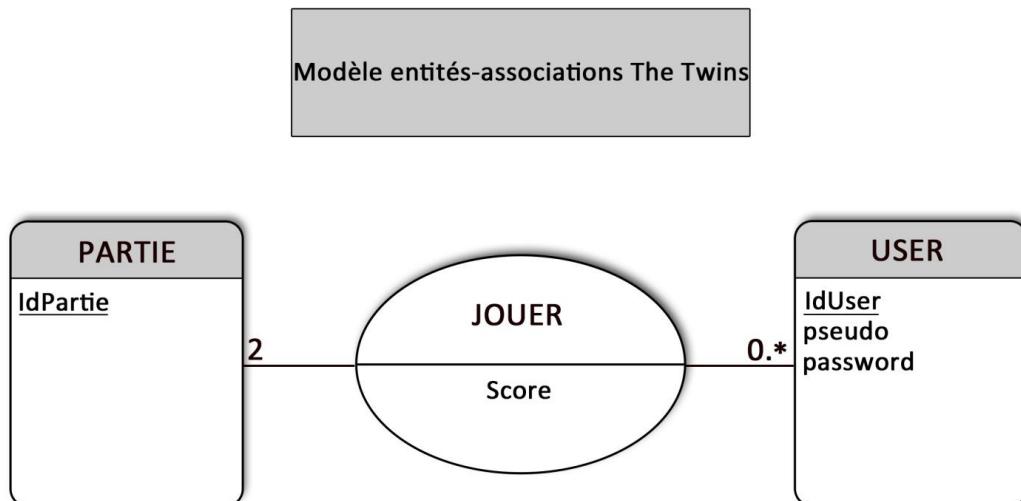
- Présenter le jeu pour ceux qui ne le connaissent pas de manière à les inciter à le télécharger. Pour cela, les internautes trouveront sur le site des images, vidéos et descriptions textuelles du jeu.
- Télécharger le manuel, la galerie ainsi que le jeu pour pouvoir y jouer.
- S'inscrire et donc créer un compte afin de pouvoir démarrer le jeu The Twins.
- Découvrir notre entreprise **Prod'IT Studio**.
- Visionner les meilleurs scores faits par les joueurs possédant un compte.

3. Conception de l'application

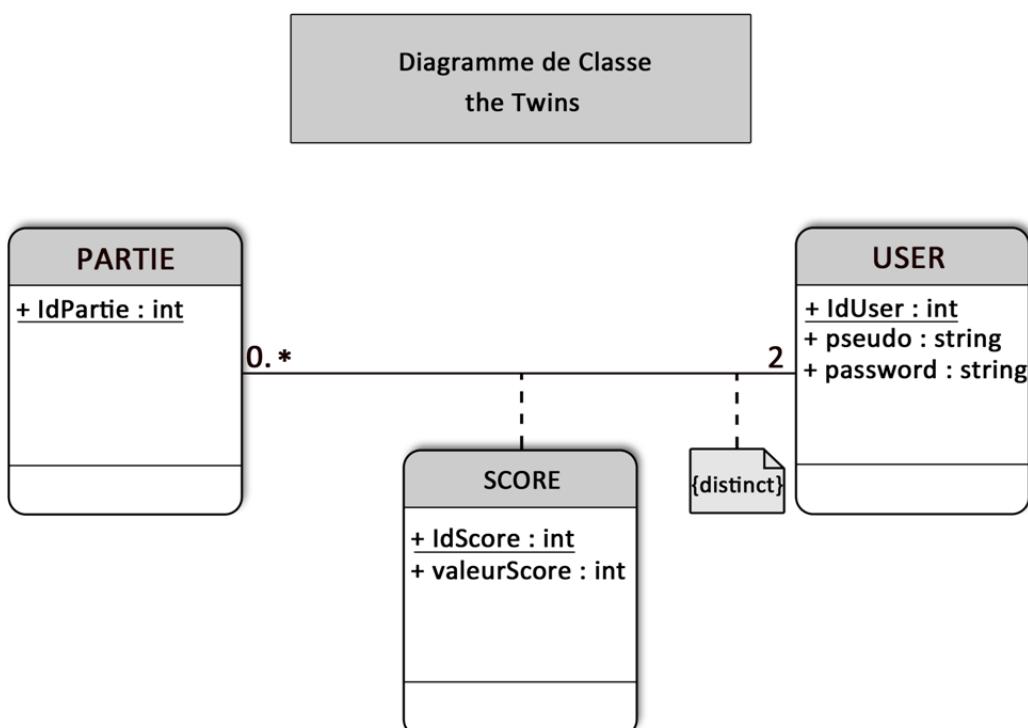
Une fois décidé sur ce que devrait faire notre application, nous avons commencé à la concevoir en utilisant encore une fois le langage UML. Nous avons séparé la conception en deux parties comme il est coutume de le faire : une conception de la vue statique et une autre de la vue dynamique.

3.1. Conception de la vue statique

3.1.1. L'exécutable effectuant à l'enregistrement des scores :



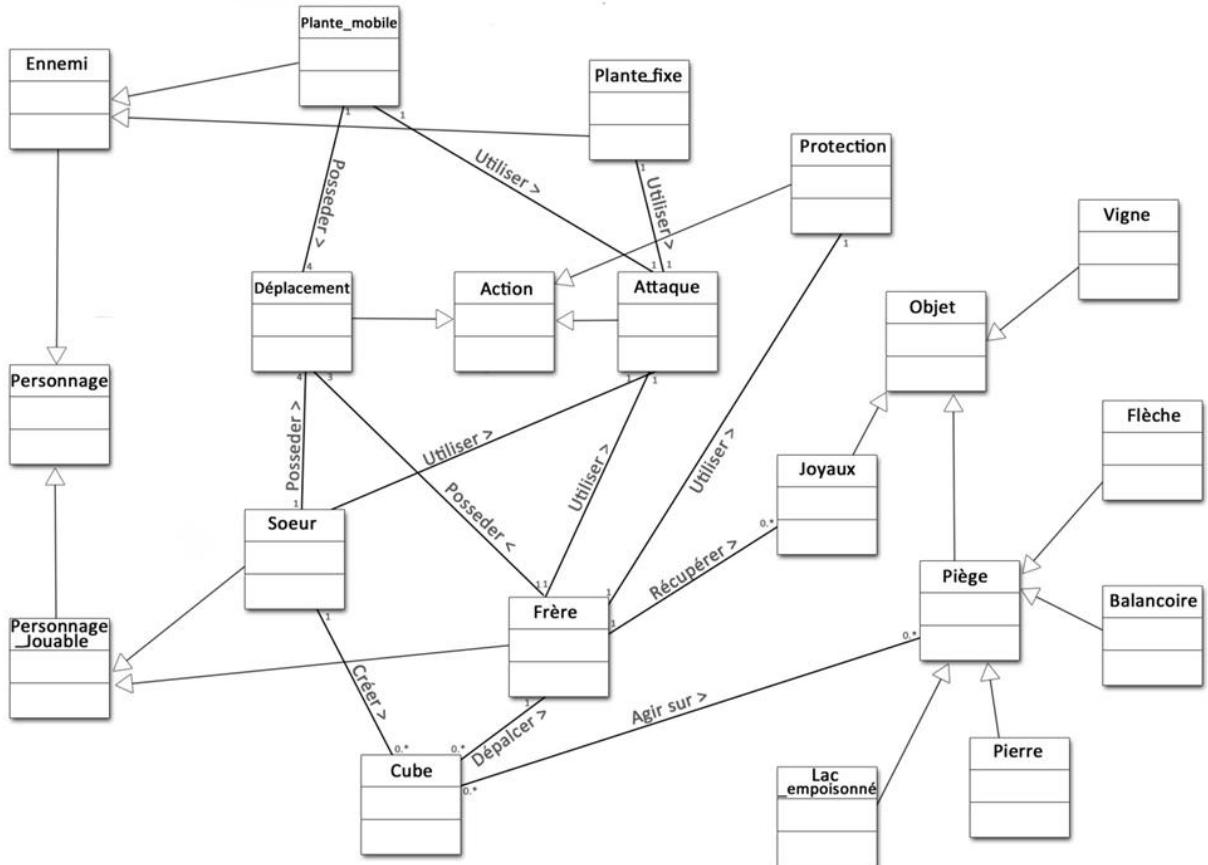
« Figure 39 : modèle entité-association The Twins »



« Figure 40 : diagramme de classe The Twins »

Ainsi, notre base de données ne comportera que trois classes utilisées pour retenir les comptes et les scores.

3.1.2. Le gameplay du jeu (liant l'exécutable du système de combat ainsi que celui des interactions avec l'environnement)



« Figure 41 : Méta-modèle d'architecture conceptuelle»

Ce modèle représente les relations structurelles entre les différents concepts de notre application. On peut remarquer deux groupes d'interactions : d'un côté, les interactions liées au combat entre les deux types de personnages et de l'autre côté, les interactions entre les personnages jouables et l'environnement.

Pour résumer brièvement nos idées :

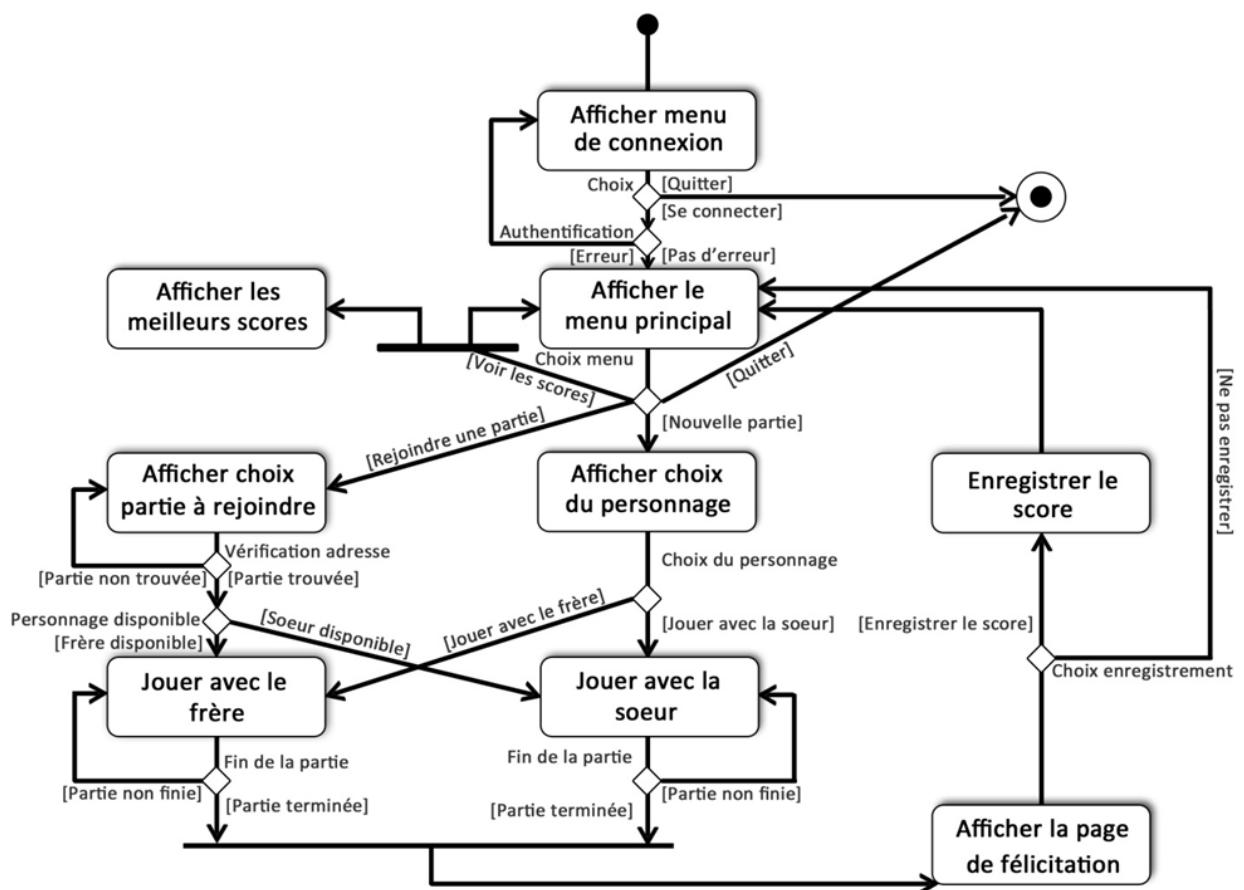
- Il y a deux grands types de personnages : ceux que les joueurs contrôlent (les personnages jouables) et ceux qui sont contrôlés automatiquement par l'ordinateur (les ennemis).
- Ces personnages peuvent réaliser des actions telles que des déplacements, des attaques ou de la protection. Ces actions composent le système de combat entre les différents personnages.

- Les personnages jouables peuvent également interagir avec les objets de l'environnement. Ces objets peuvent être neutres ou au contraire être dangereux pour les personnages et nuire à leur santé. Dans ce deuxième cas, les objets en question sont des pièges.
- Enfin, les personnages jouables peuvent s'entre-aider à l'aide des cubes, un objet créé par la sœur.

3.2. Conception de la vue dynamique

3.2.1. Conception générale de l'application

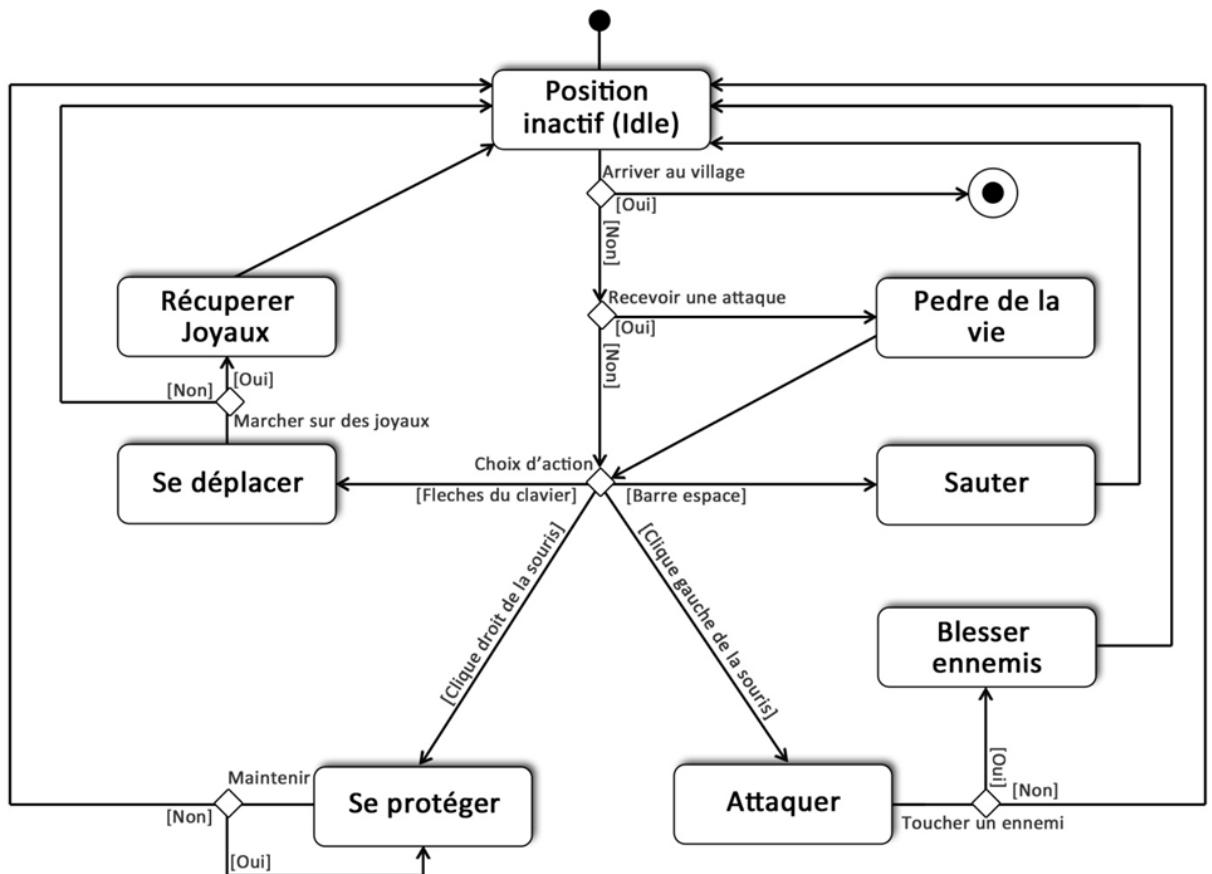
Dans ce premier diagramme d'activités, nous décrivons tous les parcours possibles d'un joueur dans l'application. On y trouve toutes les pages du jeu ainsi que les moyens d'y accéder.



« Figure 42 : diagramme d'activités de The Twins »

3.2.2. Conception du déroulement de « Jouer avec le frère »

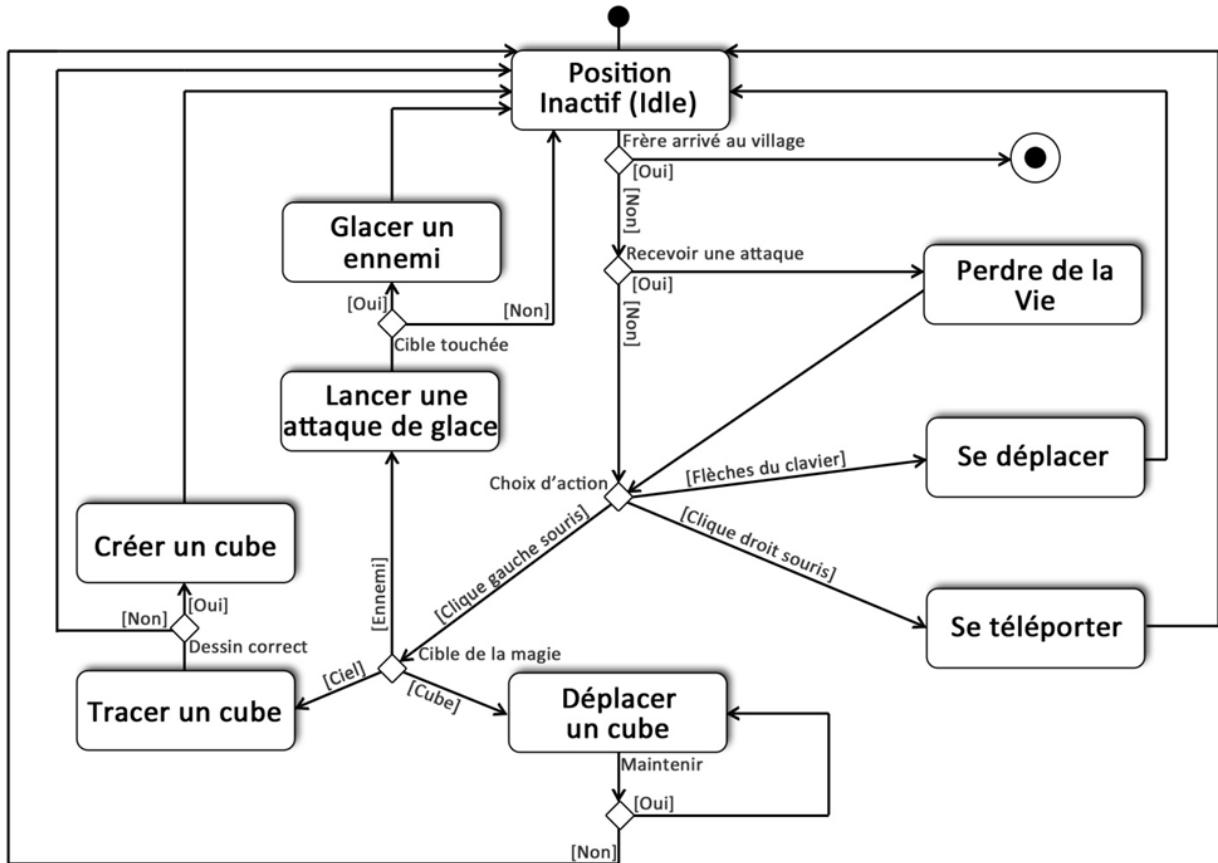
Dans ce second diagramme, nous avons détaillé le use case « Jouer avec le frère » qui correspond au **gameplay** proposé au joueur s'il joue avec le personnage « Almas ». On peut alors distinguer les différentes actions qui lui sont possibles : attaquer, se défendre, se déplacer, sauter...



« Figure 43 : diagramme d'activités pour « jouer avec le Frère » »

3.2.3. Conception détaillée de « Jouer avec la Sœur »

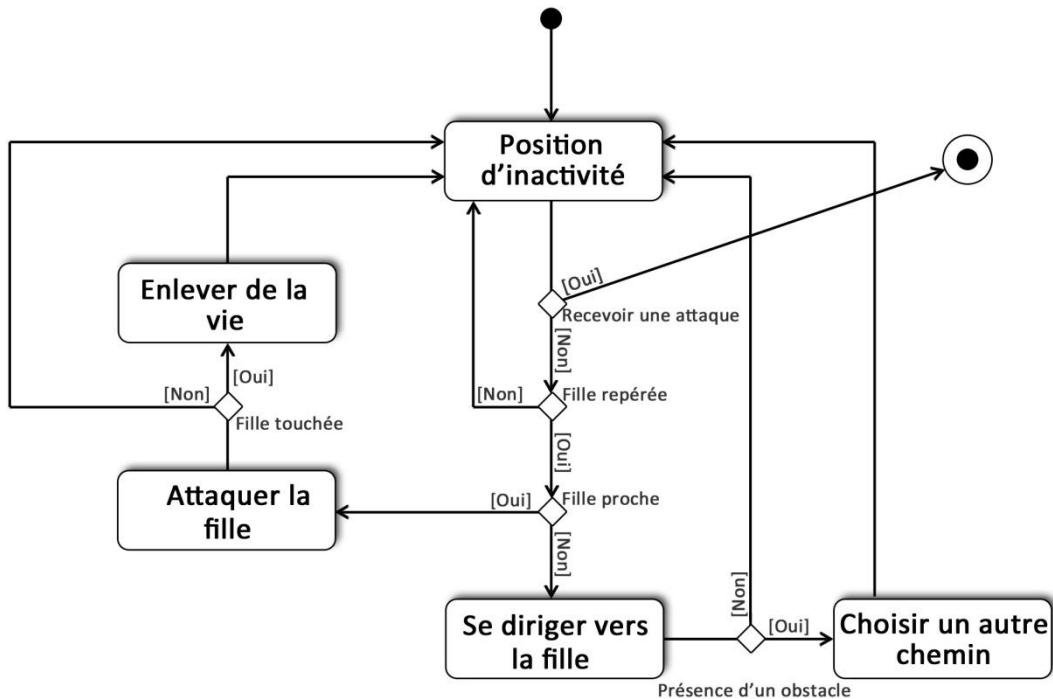
Dans ce troisième diagramme, nous expliquons le **gameplay** disponible pour le joueur contrôlant le personnage « Umi ». On peut remarquer que les fonctionnalités disponibles ne se concentrent pas sur le combat mais plutôt sur les énigmes.



« Figure 44 : diagramme d'activités pour « jouer avec la Sœur » »

3.2.4. Conception de l'intelligence artificielle

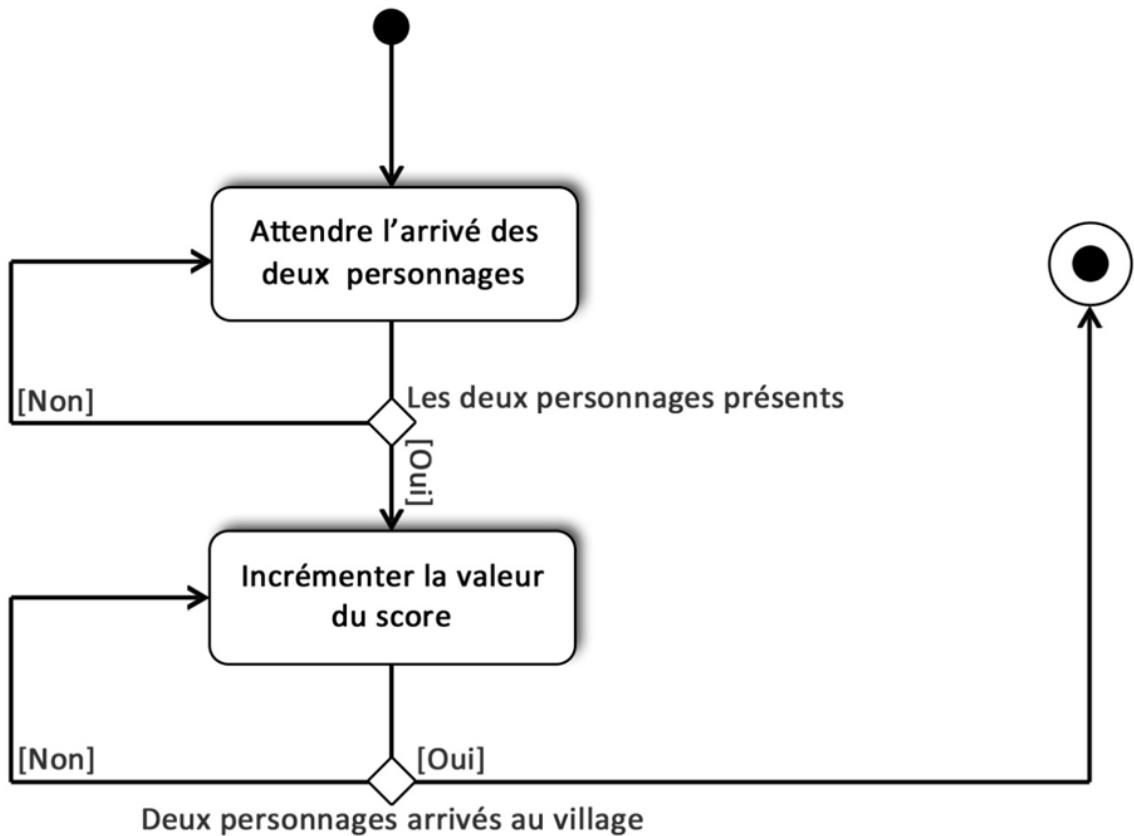
Pour que le combat puisse avoir lieu, il faut que les personnages non contrôlables puissent attaquer et se défendre. L'ordinateur doit donc les contrôler de manière intelligente. Ainsi, nous avons imaginé qu'ils devraient par exemple pouvoir détecter la cible ou éviter les obstacles présents sur leur chemin. Ce quatrième diagramme d'activités montre le fonctionnement de cette intelligence artificielle.



« Figure 45 : diagramme d'activités concernant les actions des plantes mobiles »

3.2.4. Conception du système de calcul du score

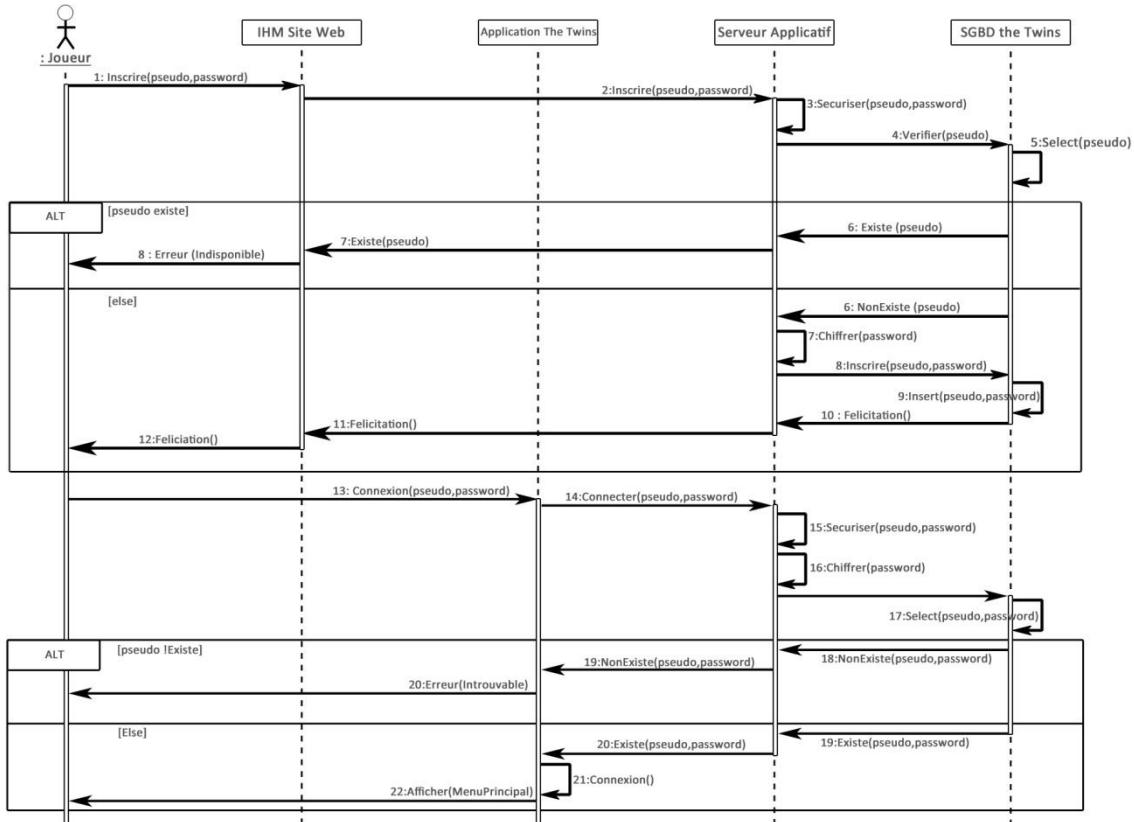
Dans ce dernier diagramme d'activités, nous avons décidé de présenter le système du temps et du score car il correspond en fait au but du jeu. On peut y voir que le temps ne commence à être incrémenter qu'après la présence des deux personnages sur la scène.



« Figure 46 : diagramme d'activités système de score »

3.2.4. Conception du déroulement du processus d'inscription et de connexion.

Notre système de calcul du score et celui du meilleur score en ligne dépendent tous les deux du compte utilisateur. L'inscription et la connexion sont donc deux éléments clés de notre projet. De plus il représente techniquement le partage d'une base de données entre un site web et un jeu 3D et en même temps l'intégration de la notion de sécurité dans notre projet. Ainsi, ils sont également au centre de notre étude. Les deux éléments étant complémentaires l'un envers l'autre, nous avons choisi de les regrouper dans un même diagramme décrivant tout le processus.



« Figure 47 : diagrammes de séquence »

4. Contraintes de qualité

Les spécifications fonctionnelles sont pratiques pour expliquer ce que devra faire l'application. Cependant, cela ne suffit pas à décrire le processus de développement qui devra être utilisé. Nous aimerais tout d'abord, en tant qu'étudiants, apprendre à travailler de façon professionnelle et ensuite nous souhaitions également que le jeu soit réellement agréable à jouer une fois fini. Ainsi, nous avons donc écrit plusieurs listes de critères que nous devrions suivre par la suite. Ces listes sont divisées en deux types :

- La qualité externe, celle qui concerne l'utilisateur final.
- La qualité interne au produit. Cette catégorie regroupe deux notions : la qualité du travail graphique et ergonomique d'une part et la qualité technique d'une autre (programmation, optimisation ...)

Pour garantir la réussite de nos attentes, nous avons également rédigé des questionnaires destinés à des joueurs pour la qualité externe (des amis à nous) mais aussi à des professionnels pour ce qui

concerne la qualité interne. Nous désirions les faire remplir une fois l'application terminée et donc testable.

4.1. Qualités internes concernant la partie technique

4.1.1. Propriété du code et organisation du travail

Une programmation ne respectant pas certaines règles, un processus non-structuré ou encore un type de programmation non-adapté sont des facteurs capables d'empêcher la maintenance, l'évolution ou l'étude d'un projet. Nous nous sommes donc résolus, sachant que notre projet serait analysé par la suite, à travailler de manière organisée.

- Réutilisabilité d'un script sur plusieurs objets.
- Utilisation de l'indentation et des commentaires des codes.
- Utilisation de l'orienté objets (classe, héritage et abstraction).
- Architecture en trois couches séparées (présentation, métier et accès aux données).
- Organisation des dossiers du projet et des objets de la scène.
- Travail avec la méthodologie « Scrum » et le découpage en tâches et catégories.
- Utilisation de « UML » pour la modélisation.

4.1.2. Optimisation de la mémoire via Unity 3D

L'une des contraintes clé du domaine du jeu vidéo est l'optimisation du calcul par le processeur et la carte graphique car la qualité logicielle est toujours bridée par les capacités des supports matériels. Plus un jeu nécessite du calcul, plus le temps de rendu d'une image sera lent. Le **FPS**³⁸, temps de calcul d'une image est très important car s'il chute en dessous de 24, le jeu devient injouable. Certains joueurs expérimentés, habitués aux jeux, sont capables de calculer visuellement le FPS et souhaitent jouer à une fréquence de, au minimum, 60FPS. Ainsi, nous devons veiller à ce que notre travail soit optimisé en calcul pour qu'un ordinateur puisse le calculer entre 24 et 60 FPS selon ses capacités.

- Utilisation de « **shader**³⁹ » de type « **Unlit**⁴⁰ » et « **VertexLit**⁴¹ » pour éviter le calcul de la projection des ombres et de la réflexion lumineuse sur les objets 3D.
- Utilisation de variables pour alléger les parcours de hiérarchie avec « **GameObject.Find**⁴² » ou « **This.GetComponent**⁴³ » qui sont longs à exécuter.
- Effets en 2D grâce à l'utilisation des « **SpriteSheets** » et des planes (à deux polygones) plutôt que de réelles modélisations comportant un grand nombre de polygones.
- Suppression des composants inutiles lors des imports des fichiers **.FBX**⁴⁴.
- Optimisation du nombre de « **Colliders**⁴⁵ » sur la scène.
- Favoriser les variables privées plutôt que les variables publiques.

³⁸ Frame Per Seconde : nombre d'images par seconde (fréquence d'affichage).

³⁹ Matériaux qu'on applique sur les objets 3D.

⁴⁰ Matériaux avec une texture qui ne réagit pas à la lumière.

⁴¹ Matériaux avec une texture illuminée par elle-même.

⁴² Une fonction qui permet de parcourir tous les objets d'une scène pour trouver l'objet désiré.

⁴³ Une fonction qui permet de parcourir tous les composants d'un objet pour trouver le composant désiré.

⁴⁴ Format d'exportation pour les objets 3D pour qu'ils deviennent compatibles avec Unity.

⁴⁵ Un composant qui permet au moteur de savoir s'il doit ou non calculer les collisions faites par un objet 3D.

4.1.3. Bonne utilisation des outils proposés par le moteur Unity et autres techniques avancées

L'intérêt d'un projet étudiant est que ces derniers apprennent des techniques avancées et utilisent celles enseignées en cours. De plus, travailler dans l'avenir avec le moteur Unity signifie connaître les outils et fonctionnalités qu'il propose afin de les utiliser de façon pertinente. Ainsi, bien que cela dépasse le cadre de la spécification et de la conception, nous avons listé les outils et techniques que nous souhaitons voir présente dans le jeu :

- Utilisation des composants « **Physics**⁴⁶ » et « **Rigidbody**⁴⁷ » pour les énigmes et contrôles des personnages. Deux outils permettant de gérer la physique dans le jeu.
- Effets spéciaux et effets d'écran à l'aide de la classe « **Renderer**⁴⁸ ».
- Utilisation de « **Shuriken**⁴⁹ », le nouveau système de particules. Plus puissant et plus optimisé, Shuriken permet de créer des effets tels que de la magie de glace ou de la poussière.
- Utilisation de « **Mecanim**⁵⁰ » et du composant « **Animator**⁵¹ ». Plus puissant que le simple composant « Animation », Mecanim permet notamment de créer des fusions entre les animations.
- Mise en multi-joueurs sur une même partie. Cela implique de gérer beaucoup d'outils et classes de Unity tels que les « **Network**⁵² », les « **NetworkView**⁵³ » ou les réglages du « **Send Rate**⁵⁴ »...
- Connexion via une base de données hébergée sur un serveur payant à partir, à la fois, du jeu et du site web. Cela sous-entend la maîtrise de la classe Unity « **WWW**⁵⁵ ».
- Sécurité de la base de données en chiffrant les mots de passe.
- Intelligence artificielle des ennemis (détection des obstacles à éviter).
- Créations de pouvoirs magiques et maîtrise des mouvements de souris.

4.2. Qualités internes concernant la partie graphique et ergonomique

Le travail graphique d'un jeu vidéo se découpe en cinq principales tâches :

- la conception des « **assets**⁵⁶ » (personnages, décors, et effets visuels) par l'« artiste de conception » qui travaille avec des logiciels de 2D tels que « Adobe Photoshop » ou parfois même sur papier.
- La création des objets en 3D par un « modélisateur / sculpteur » qui utilise des logiciels tels que « Autodesk Maya », « 3DS Max » et « Zbrush » et se base sur les conceptions 2D faites précédemment (appelées aussi « **blueprints**⁵⁷ »).

⁴⁶ Une classe qui concerne la physique.

⁴⁷ Une classe qui concerne les objets rigides (qui ont une masse et qui occupe un volume sur la scène).

⁴⁸ Une classe qui gère le rendu des objets 3D sur la scène.

⁴⁹ Le nouveau système de particules de Unity.

⁵⁰ Une fonctionnalité de Unity qui permet la gestion avancée d'animations.

⁵¹ Une classe qui concerne des animations.

⁵² Une classe qui concerne les fonctionnalités de type réseau.

⁵³ Une classe qui concerne les fonctionnalités de type réseau.

⁵⁴ Taux d'envoi.

⁵⁵ Une classe qui concerne les fonctionnalités web.

⁵⁶ L'ensemble des modélisations 3D qui concernent l'application.

⁵⁷ Plan de références aidant à modéliser fidèlement un personnage à la base dessiné.

- L'animation des modélisations 3D et des effets visuels (par exemple les « SpriteSheets ») créés précédemment par un animateur 2D / 3D.
- Ensuite, le « Level Design », la dernière étape, est l'assemblage fait des objets 3D pour créer l'environnement final du jeu. Le « Level Designer » travaille donc tel un architecte et place les objets. Il compose soit directement dans le moteur graphique du jeu ou dans le logiciel de modélisation 3D et importe la scène dans le moteur.
- Enfin, le « designer d'interface » est celui qui, tout comme dans les métiers du web ou des applications, crée les menus, icônes et boutons.

Nous avons essayé de créer un travail graphique respectant ces différentes étapes et avons établi des critères de qualité pour la partie 2D ainsi que pour la partie 3D.

4.2.1. Qualité du travail 2D

- Création de « **sketches**⁵⁸ » et « **blueprints** » (images de références contenant les vues de face et de profil en 2D pour les futures modélisations 3D des personnages) beaux, respectant les proportions du style choisi et cohérents les uns avec les autres.
- « **Texturing**⁵⁹ » de qualité ce qui inclut un découpage optimisé des UV (pas de « **stitching**⁶⁰ » ni de perte d'espace inutile sur le « **0-1 space**⁶¹ » des UV), une peinture réussie (cohérence des couleurs et souci du détail).
- « **SpriteSheets** » et systèmes de particules convaincants et apportant un ajout bénéfique visuellement. Par exemple, dans un jeu, la sensation de frappe est créée par l'ajout d'effets visuels tels que les explosions ou les ondes de choc.
- Création de menus et d'interfaces utilisateur ergonomiques, offrant une lisibilité confortable, une visibilité et une cohérence graphique avec le monde 3D.

4.2.2. Qualités du travail 3D

- Fidélité des modélisations par rapport à leurs « **blueprints** » respectifs.
- Optimisation du nombre de polygones utilisé lors de la modélisation 3D pour éviter un temps de calcul trop long dans le jeu (rendu en temps réel).
- Création d'une « **topologie** » permettant l'animation. Afin qu'un objet 3D puisse être animé, le flux des polygones qui le composent doivent respecter certaines règles telles que le fait que chaque polygone de l'objet doit contenir 3 ou 4 sommets uniquement (des triangles ou des carrés), que les zones articulées lors de l'animation sont composées de plusieurs rangés de polygones au lieu d'une seule et que la linéarité d'une ligne de polygones ne soit pas rompue.
- Bonne maîtrise des outils tels que « **Unwrap UVW**⁶² » de 3DS Max pour le découpage des UV.
- Assurer le « **rigging**⁶³ » (mise en place d'un squelette composé de « **bones** » manipulés à l'aide de « **controllers**⁶⁴ ») et le « **skinning**⁶⁵ » (phase de lien entre le squelette et les polygones du personnage 3D pour pouvoir animer ce dernier).

⁵⁸ Croquis, dessins.

⁵⁹ Le fait de peindre les modélisations 3D.

⁶⁰ Étirement et déformation de peinture.

⁶¹ L'espace dans lequel on peut peindre sur les UV.

⁶² Un modificateur de 3DS Max qui permet le découpage des UV.

⁶³ Le processus de création de squelette d'articulations pour les modélisations 3D.

- Création d'animations respectant les règles du jeu vidéo. Pour que l'objet puisse être utilisé dans un jeu, il faut que ses animations soient conçues sans le déplacer (pour éviter les problèmes de « collider », l'objet sera déplacé à l'aide de la programmation). Il faut également que les animations puissent être répétées sans coupures (« **loops**⁶⁶ ») et qu'elles soient visuellement crédibles (le joueur doit comprendre l'animation que le personnage est en train de faire).

4.3. Qualités externes, vues par l'utilisateur

Pour la charte de la qualité vue par l'utilisateur, il nous a paru plus adéquat de la rédiger sous la forme d'un questionnaire. Ainsi une fois le développement du jeu terminé et une fois testé, nous connaîtrons la valeur réelle de notre application pour les joueurs.

Nous avons considéré que l'intérêt d'un joueur pour un jeu suit en quelques sortes une procédure chronologique, identique pour tous. Chaque étape de ce processus a besoin, pour être réalisée, que l'étape précédente fut réussie.

- Quand l'utilisateur voit le jeu pour la première fois, dans une bande annonce par exemple, il faut qu'il soit impressionné ou qu'il ait apprécié ce qu'il a vu pour qu'il ait envie de le tester. Ceci est la première phase que nous avons appelée : l'« accroche sensorielle ».
- Une fois qu'il décide de tester le jeu, il faut que la maniabilité du jeu et que les fonctionnalités qui lui sont proposées rendent le jeu agréable et, si possible, qu'ils apportent une nouveauté. Cette phase est la « Jouabilité ».
- Même le jeu possédant le gameplay le plus abouti finit par lasser les joueurs avec le temps. Il faut, pour retarder cette lassitude, que le jeu possède une histoire qui pousse le joueur à vouloir avancer et connaître la suite (éveil de la curiosité, intrigue). Il faut aussi qu'une fois le jeu terminé, le joueur ait la sensation d'avoir vécu une aventure ou une expérience. Cette phase est en rapport avec la scénarisation.

4.3.1. Accroche sensorielle :

- *Composition graphique :*
 - Comment trouvez-vous les menus ?
 - Les menus donnent-ils une idée sur l'univers du jeu ?
 - Sont-ils ergonomiques ?
 - Vous êtes sur la première scène du jeu. Quelle était votre première réaction ?
 - Après avoir visité tout le niveau, avez-vous trouvé que le jeu est beau ?
 - L'univers du jeu est supposé être magique. Cette ambiance y était ?
- *Composition sonore :*
 - Est-ce que la musique du jeu a contribué à la création de cette ambiance magique ?

4.3.2. Jouabilité :

- *Satisfaction du gameplay :*

⁶⁴ Des objets qui permettent d'articuler le squelette manuellement.

⁶⁵ Le processus de lier les polygones d'une modélisation 3D à son squelette.

⁶⁶ Boucles.

- Le contrôle des personnages vous a été fluide ? Les personnages répondaient-ils à vos commandes ?
 - Vous vous êtes amusés à y jouer ?
 - Avez-vous trouvé les énigmes intéressantes ?
 - Avez-vous trouvé les combats intéressants ?
 - Y avait-il du défi ?
- *Satisfaction sociale* :
 - Avez-vous aimé l'idée de jouer en coopération ?
 - Pensez-vous que créer une page communautaire est une bonne idée ?
 - Trouvez-vous l'esprit compétitif dans ce jeu grâce au système de score en ligne ?
 - *Originalité* :
 - Quelle était la chose que vous avez aimée le plus dans le jeu ?

4.3.3. Scénarisation :

- Avez-vous aimé l'histoire ?
- Pour un seul niveau, trouvez-vous que la durée de vie est satisfaisante ?



L'aventure
commence

À présent, décidés sur ce que nous allions créer et très impatients de commencer, le temps était venu pour nous de démarrer nos logiciels, « *Que l'aventure commence !* ».

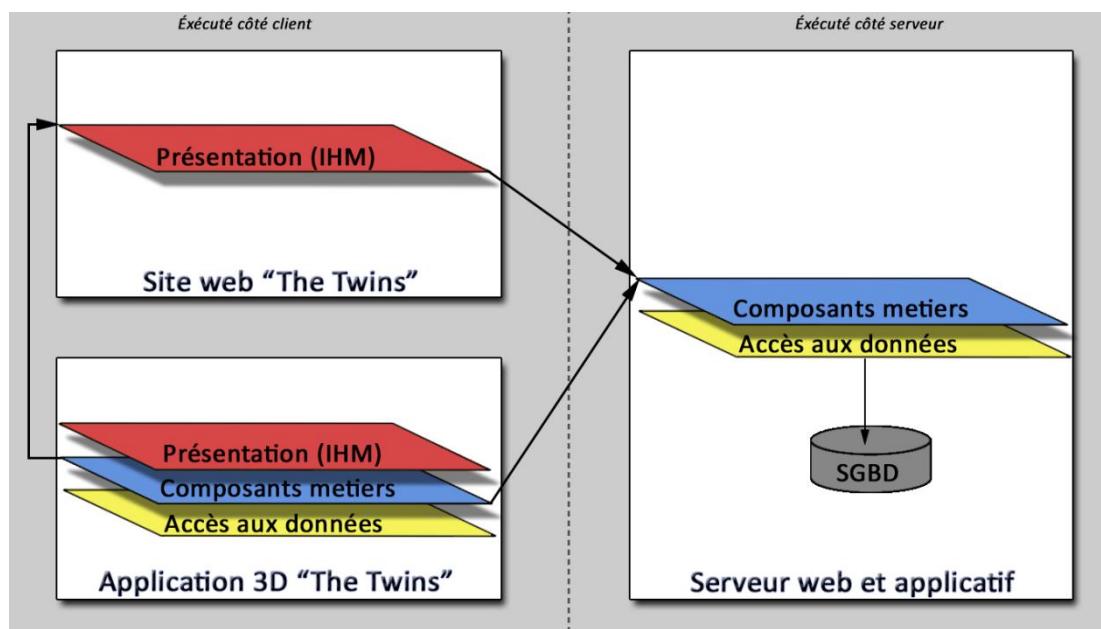
1. Déroulement du projet et méthode de travail

1.1. Organisation des dossiers du projet et langage choisi

Nos applications respectent une architecture en 3 couches. Le site web est une application client « **léger** » et donc, seule la partie « **IHM** » est calculée chez l'utilisateur. Le jeu, au contraire, est une application client « **lourd** ». Certaines données sont enregistrées localement et c'est également le cas de la majorité des traitements de la couche « **métier** ».

La couche métier exécutée chez le client regroupe différents composants tels que l'intelligence artificielle, la connexion multi-joueurs ou encore le système de calcul du score. Ces différents composants sont les livrables produits par les « **sprints** ».

La couche métier exécutée sur le serveur permet l'interaction avec la base de données. On y trouve donc des traitements tels que l'inscription, la connexion ou encore la récupération des meilleurs scores.



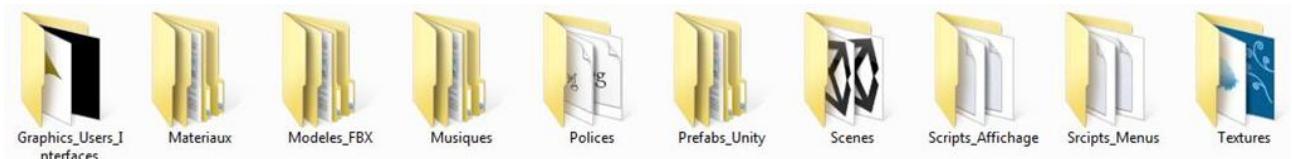
« Figure 48 : Les couches de nos applications »

Pour respecter cette architecture en trois couches, nous avons commencé par ranger les dossiers du jeu de manière à séparer ces différentes couches [voir figure 49] :



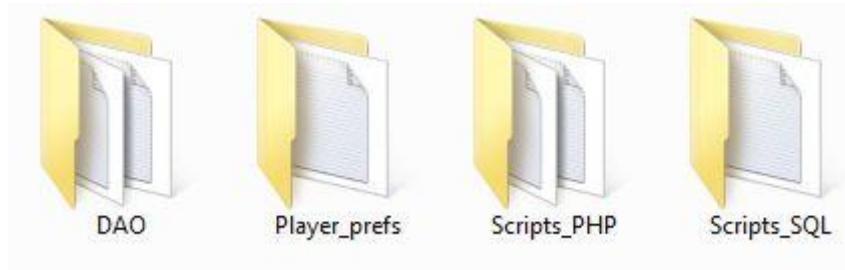
« Figure 49 : les dossiers du projet »

- Le dossier « **COUCHE_PRESENTATION** » contient les objets 3D, les textures, les matériaux, les scènes Unity ou encore les scripts UnityScript (basé sur la syntaxe de JavaScript mais avec la présence d'un « FrameWork » complet propre à l'utilisation de Unity) concernant l'affichage et les interfaces [voir figure 50].



« Figure 50 : contenu du dossier COUCHE_PRESENTATION »

- Le dossier « **COUCHE_DONNEES** » contient les scripts PHP5 / SQL de la base de données et ses accès, des scripts UnityScript pour y accéder à partir du jeu ou encore des scripts utilisant la classe « PlayerPrefs » de Unity pour l'enregistrement en local [voir figure 51].



« Figure 51 : contenu du dossier COUCHE_DONNEES »

- Enfin, le dossier « **COUCHE_METIER** » ne contient que des scripts UnityScript. Certains correspondent à l'intelligence artificielle ou aux scripts de contrôle des personnages (incrément « Système de combat »), d'autres mettent en relation les scripts d'affichage de la couche présentation avec ceux de la couche d'accès aux données (incrément «Création des menus»), d'autres encore permettent la mise en multi-joueurs du jeu et enfin d'autres créent les interactions possibles entre les personnages et les objets de l'environnement qui les entourent (incrément «Création de l'environnement») [voir figure 52].

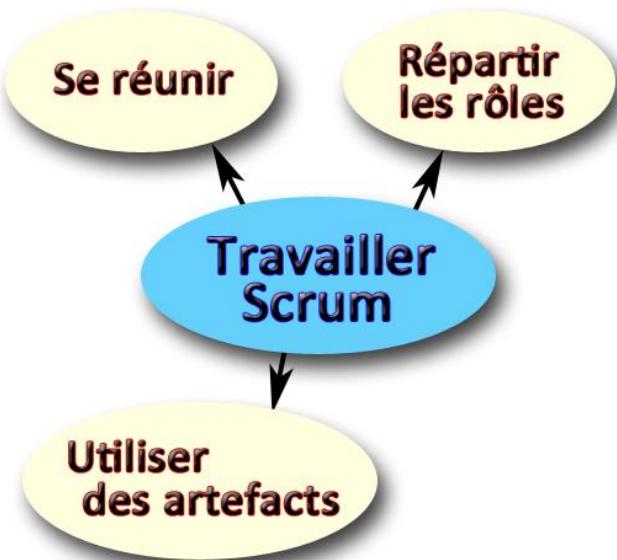


« Figure 52 : contenu du dossier COUCHE_METIER »

1.2. Le travail avec Scrum et méthode d'encadrement durant le projet

Travailler avec scrum signifie respecter trois principales règles :

- Utiliser les artefacts tels que le tableau d'avancement des tâches, le découpage appelé « **backlog** » du projet, ainsi que les « **backlog** » de chaque sprints.
- Respecter les différentes réunions de scrum : la mêlée quotidienne, les planifications et revues de sprints ainsi que les réunions de rétrospectives.
- Répartir les rôles : le « **Scrum Master** » qui vérifie le respect des artefacts et réunions, le « **Product Owner** » qui fait le backlog et enfin, l'équipe de développeurs.



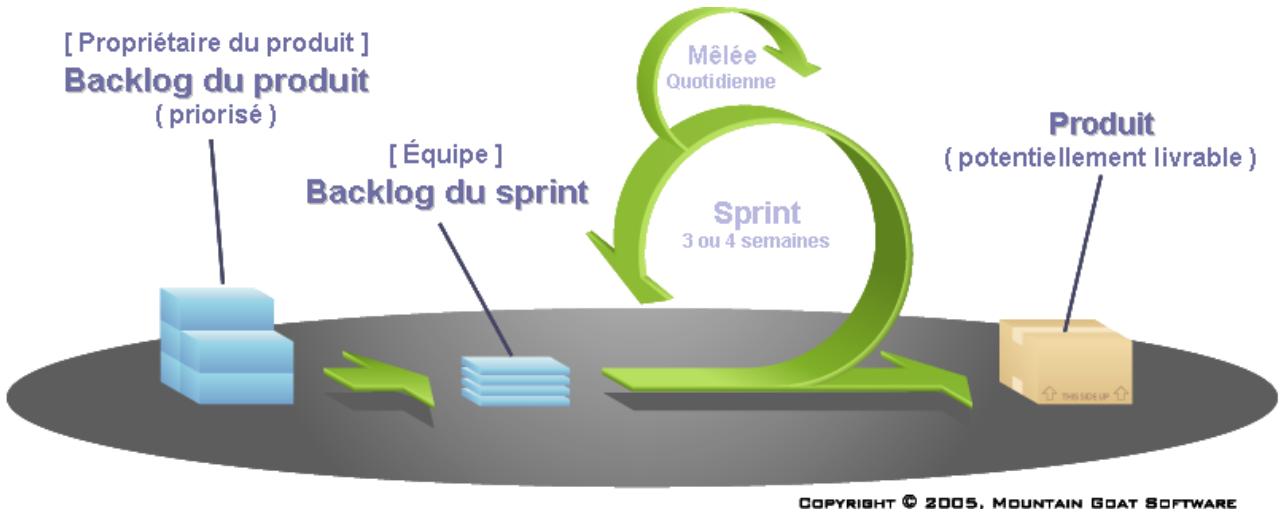
« Figure 53 : Les éléments clés de scrum »

L'encadrement avec Madame Jihene Malek se faisait une fois par semaine, mais elle joua en plus une aide importante quotidiennement sur « Trello » [voir figure 49] dans la vérification de l'avancement. Nous pouvions y envoyer nos fichiers, créer le tableau de tâches de « scrum » à l'aide des « cards » (cartes) ou encore créer une discussion avec elle. Ces discussions représentaient notamment les réunions de scrum.

Nous avons réparti les rôles scrum de la manière suivante :

- L'« **équipe** » qui travaille chaque jour, commence sa journée par une mêlée quotidienne

- de mise au point, effectue le « **backlog du sprint**⁶⁷ » et signale son avancement grâce au tableau de tâches. Équipe constituée de Achref Bouhadida et Anas Neumann.
- Le « **Scrum Master**⁶⁸ », personne qui vérifie le respect des réunions et l'avancement sur le tableau de tâches : Madame Jihen Malek, notre encadrante.
 - Le « **Product Owner**⁶⁹ », personne découpant le projet en tâches (ou « **User Stories** ») et en livrables. Il effectue donc ce que l'on appelle le « **Backlog du produit**⁷⁰ ». Dans notre cas, le découpage a été fait à deux (Anas Neumann et Achref Bouhadida).
- [voir figure 54]



1.3. Gestion du Tableau de tâches de scrum

Pour contrôler l'avancement, nous avons créé trois principales listes de tâches :

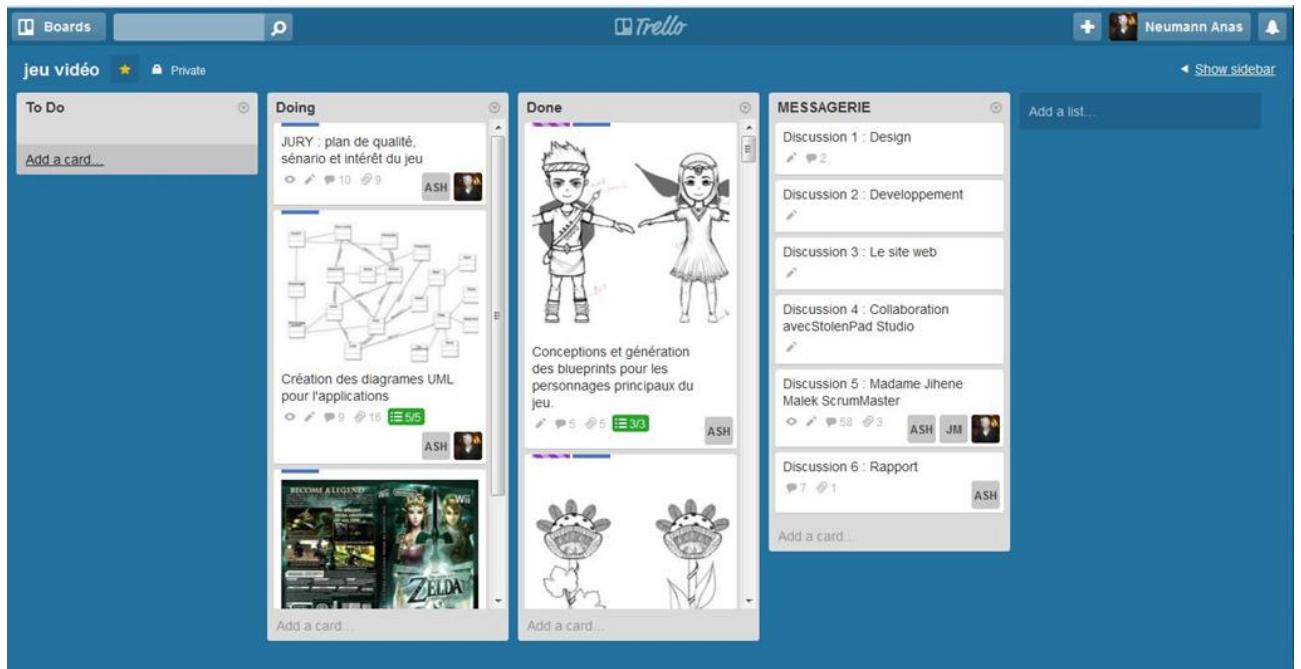
- « **To do** », catégorie pleine au début et totalement vide à la fin, « To do » contient toutes les tâches qui n'ont ni été réalisées ni même commencées.
- « **Doing** », catégorie dans laquelle on affichait les tâches en cours de réalisation. Une tâche restait dans cette liste tant qu'elle n'était pas terminée.
- « **Done** ». Cette dernière catégorie, vide au début et pleine à la fin, comprend toutes les tâches terminées complètement.

⁶⁷ Toutes les tâches qui concernent un seul sprint.

⁶⁸ Maître de la mêlée.

⁶⁹ Possesseur du produit.

⁷⁰ Tous les sprints du produit.



« Figure 55 : Trello »

« Trello » fournissait des fonctionnalités sur les tâches permettant un contrôle et une organisation en profondeur de notre travail telles que :

- La mise en place d'une liste de catégorie de tâches appelée « **Label** ». Cette fonctionnalité a permis de regrouper les tâches selon leurs emplacements dans l'application.
- Une fonctionnalité appelée « **Members** » permettant de spécifier l'étudiant qui allait réaliser la tâche.
- La mise en place d'une sous liste d'étapes à l'intérieur même d'une tâche. Ainsi, nous pouvions évaluer le temps restant pour une tâche dans la liste « Doing ». Cette fonctionnalité se nomme « **CheckList** ».
- La capacité d'ajouter des fichiers ou des commentaires dans une tâche (« card »).

2. Backlog du produit et plan des releases

2.1. Backlog du produit

La priorité d'une tâche est souvent définie selon un calcul tenant compte de sa complexité, durée ou encore nombre de personnes qui travaillent dessus. Cependant, nous avons décidé de simplifier le processus en se limitant uniquement à l'ordre dans lequel les tâches doivent être réalisées. Par exemple si une tâche doit être réalisée avant une autre, elle est plus prioritaire. Nous avons également défini les priorités en respectant la règle suivante : une priorité plus élevée possède une valeur plus basse. Ainsi la première tâche du projet à réaliser possède la priorité « 1 ».

Nom de la tâche	Priorité
Création du projet Unity, des dossiers et de l'environnement de travail.	1
Conception des personnages principaux	2
Conception des deux ennemis	3
Modélisation 3D du garçon	4
Modélisation 3D de la fille	5
Modélisation 3D de la plante d'attaques à distance	6
Modélisation 3D de la plante d'attaques rapprochée	7
Découpage UV de la fille	8
Découpage UV du garçon	9
Découpage UV de la plante d'attaques à distance	10
Découpage UV de la plante d'attaques rapprochée	11
Rigging et Skinning de la fille	12
Rigging et Skinning du garçon	13
Rigging et Skinning de la plante d'attaques à distance	14
Rigging et Skinning de la plante d'attaques rapprochées	15
Animation du garçon	16
Animation de la fille	17
Animation des deux plantes	18
Création des papillons	19
Scripts de connexion sécurisée sur un compte Prod'IT et base de données SQL	20
Scripts de l'envoi du score en ligne et sa visualisation	21
Scripts pour la connexion en multi-joueurs	22
Scripts de contrôle des caméras	23
Gestion Mecanim du garçon	24
Gestion Mecanim de la fille	25
Scripts pour le contrôle du garçon	26
Scripts pour le contrôle de la fille	27
Gestion des animations de la plante d'attaques à distances	28
Gestion des animations de la plante d'attaques rapprochée	29
Scripts de l'intelligence artificielle de la plante d'attaques à distance	30

Scripts de l'intelligence artificielle de la plante d'attaques rapprochées	31
Création des systèmes de particules et objets pour les pouvoirs de la fille	32
Création des effets pour l'épée et les pas du garçon	33
Création des particules et effets pour les ennemis	34
Ajout des effets et systèmes de particules dans les scripts	35
Rédaction du scénario	36
Conception de l'environnement et objets du décor	37
Conception des objets « modulables »	38
Modélisation 3D des objets du décor	39
Modélisation 3D des objets « modulables »	40
Textures et UV des objets « modulables »	41
Textures et UV des objets de décor	42
Conception des énigmes	43
Modélisation des objets d'énigmes	45
Textures et UV des objets d'énigmes	46
Level Design et composition finale de la scène	47
Scripts de déroulement du temps et du score	48
Délimitation de l'espace de jeu	49
Scripts de récupération des diamants	50
Scripts de collision avec les eaux	51
Scripts de l'énigme des ponts	52
Scripts de l'énigme de la balançoire	53
Scripts des plateformes mobiles	54
Scripts des flèches	55
Scripts de la zone d'éboulement du rocher	56
Scripts de la zone des blocs de pierres balançoires	57
Scripts de la zone de fin de niveau	58
Ajout des effets de particules (eau, poison, feu...)	59
Conception des GUI	60
Peinture des GUI dans tous leurs états	61
Conception des pages du jeu	62

Conception des maquettes des pages	63
Conception du schéma de navigation entre les pages	64
Création des fonds et du logo du jeu	65
Création du manuel d'utilisation du jeu	67
Peinture des éléments des pages	68
Création d'un curseur	69
Développement des GUI	70
Développement des pages de menus	71
Création des effets entre les pages	72
Développement de la page des meilleurs scores et fin de partie	73
Modification de la base de données pour les scores	74
Création des scripts PHP d'accès et d'ajout de meilleurs scores	75
Création des scripts d'interaction avec le jeu en JavaScript	76
Choix des formes et couleurs du site web	77
Logo de la société Prod'IT Studio	78
Conception schéma de navigation du site web	79
Développement HMTL	80
Intégration CSS et JavaScript	81
Développement page PHP d'inscription	82
Rédaction du contenu	83
Intégration d'une vidéo du jeu	84
Création de la page Facebook, Google+ et Youtube de Prod'IT Studio	85
Hébergement en ligne du site web	86
Ajout d'effets visuels et de caméra	87
Ajout de la musique dans le jeu	88
Réglages de compilation du projet (icône, qualité...)	89
Compilation finale et hébergement sur le site	90
Rédaction des tests pour les utilisateurs	91
Phases de test et feedback	92

« Tableau 1 : backlog du produit »

2.2. Plan des release

Après avoir découpé notre projet final en tâches, il nous a fallu regrouper ces tâches en sprints réalisables dans un temps compris entre deux et quatre semaines. Un sprint, bien qu'étant une partie incomplète du projet doit être un livrable exécutable et réutilisable. Il faut donc délimiter les sprints de manière à respecter cette règle. Après le regroupement, nous avons obtenu 5 sprints à réaliser :

- ***Sprint 1 : Le système de combat multi-joueurs et connexion en ligne.*** Un livrable avec lequel deux joueurs peuvent se connecter à une base de données avec leurs pseudos et jouer à deux en réseau. Avec ce jeu, les personnages peuvent combattre et se déplacer. (Une durée de quatre semaines durant le mois de février.)
- ***Sprint 2 : L'environnement et les énigmes, le monde des Twins.*** Crédation d'un monde forestier parsemé de pièges et d'énigmes à résoudre. Avec ce jeu, les joueurs ne peuvent que visionner le monde mais en le combinant avec le sprint précédent, le joueur obtient un jeu dans lequel il peut se déplacer, se battre et résoudre des énigmes. Les joueurs pourront alors faire un score et l'enregistrer dans la base de données. (Une durée de quatre semaines durant le mois de mars.)
- ***Sprint 3 : L'interface du jeu.*** Crédation d'une application possédant des menus permettant au joueur de démarrer les deux premiers sprints et ajout dans la partie de jeu les indicateurs « GUI ». Ce sprint comprend également la création du logo et de fond liés au jeu. Ainsi, Le jeu est complet. (Une durée de quatre semaines durant le moi d'avril.)
- ***Sprint 4 : Prod'IT Studio.*** Crédation du site web du jeu, de la page Facebook, Google+ et Youtube de la société Prod'IT Studio. La tâche inclut la création du logo de la société. (Une durée de deux semaines durant le mois de mai.)
- ***Sprint 5 : Déploiement et tests finaux.*** Export du jeu en exécutable, mise en ligne sur le site web et création d'une bande annonce. À ce stade, le projet est fini. (Une durée de deux semaines durant le mois de mai.)

Nous allons dès maintenant vous présenter la réalisation de chaque sprint du jeu dans l'ordre en débutant chaque partie par un tableau prévisionnel.

I. Sprint 1 :

Système de combat multi-joueurs et connexion en ligne



1. Tableau prévisionnel du sprint

- *Date de début du sprint* : Lundi 03 Février 2014
- *Date de fin de sprint* : Vendredi 28 Février 2014
- *Temps estimé en heures* : 260 heures
- *Échelle de mesure* :
 - Une journée égale 8 heures de travail (de 9h->13h puis de 14h->18h)
 - On compte dans le tableau, les heures restantes en fin de journée (donc après 8 heures de travail).
- *Objectifs du sprint* :
 - Création des deux personnages principaux et de leurs ennemis. Création du système de combat qui confronte les personnages jouables et les intelligences artificielles.
 - Mise en place du réseau dans le jeu pour pouvoir se connecter sur son compte Prod'IT et jouer en mode « Multijoueurs ».

1.1. Première partie du mois :

Nom :	Priorité	Temps	3	4	5	6	7	8	9	10	11	12	13	14	15
Conception des personnages principaux	2	16h	8	0	0	0	0	0	0	0	0	0	0	0	0
Conception des deux ennemis	3	16h	16	16	16	16	16	16	16	16	16	16	8	8	8
Modélisation 3D du garçon	4	8h	8	8	0	0	0	0	0	0	0	0	0	0	0
Modélisation 3D de la fille	5	8h	8	8	8	0	0	0	0	0	0	0	0	0	0
Modélisation 3D de la plante d'attaques à distance	6	8h	8	8	8	8	8	8	8	8	8	8	8	0	0
Modélisation 3D de la plante d'attaques rapprochée	7	8h	8	8	8	8	8	8	8	8	8	8	8	8	0
Découpage UV de la fille	8	8h	8	8	8	8	8	0	0	0	0	0	0	0	0
Découpage UV du garçon	9	8h	8	8	8	8	0	0	0	0	0	0	0	0	0
Découpage UV de la plante d'attaques à distance	10	4h	4	4	4	4	4	4	4	4	4	4	4	4	4

Découpage UV de la plante d'attaques rapprochée	11	4h	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
Rigging et Skinning de la fille	12	8h	8	8	8	8	8	8	8	8	8	0	0	0	0	0	0
Rigging et Skinning du garçon	13	8h	8	8	8	8	8	8	8	0	0	0	0	0	0	0	0
Rigging et Skinning de la plante d'attaques à distance	14	4h	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
Rigging et Skinning de la plante d'attaques rapprochée	15	4h	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
Animation du garçon	16	8h	8	8	8	8	8	8	8	8	0	0	0	0	0	0	0
Animation de la fille	17	8h	8	8	8	8	8	8	8	8	8	8	0	0	0	0	0
Animation des deux plantes	18	4h	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
Création des papillons	19	8h	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
Scripts de connexion sécurisée sur un compte Prod'IT et base de données	20	4h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Scripts de l'envoi du score en ligne et sa visualisation	21	8h	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Scripts pour ma connexion en multi-joueurs	22	8h	8	4	0	0	0	0	0	0	0	0	0	0	0	0	0
Scripts de contrôle des caméras	23	4h	4	4	4	0	0	0	0	0	0	0	0	0	0	0	0
Gestion Mecanim du garçon	24	4h	4	4	4	4	4	4	4	4	4	4	4	4	0	0	0
Gestion Mecanim de la fille	25	4h	4	4	4	4	4	4	4	4	4	4	4	4	0	0	0
Scripts pour le contrôle du garçon	26	24h	24	24	24	24	24	24	24	24	24	24	24	24	24	24	16

« Tableau 2 : première partie du mois »

1.2. Deuxième partie du mois :

Nom :	Priorité	Temps	16	17	18	19	20	21	22	23	24	25	26	27
Rigging et Skinning de la plante d'attaques à distance	14	4h	0	0	0	0	0	0	0	0	0	0	0	0
Rigging et Skinning de la plante d'attaques rapprochée	15	4h	0	0	0	0	0	0	0	0	0	0	0	0
Animation du garçon	16	8h	0	0	0	0	0	0	0	0	0	0	0	0
Animation de la fille	17	8h	0	0	0	0	0	0	0	0	0	0	0	0
Animation des deux plantes	18	8h	8	0	0	0	0	0	0	0	0	0	0	0

Création des papillons	19	8h	8	8	0	0	0	0	0	0	0	0	0	0	0	0	0
Scripts de connexion sécurisée sur un compte Prod'IT et base de données SQL	20	4h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Scripts de l'envoi du score en ligne et sa visualisation	21	8h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Scripts pour la connexion en multi-joueurs	22	8h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Scripts de contrôle des caméras	23	4h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Gestion Mecanim du garçon	24	4h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Gestion Mecanim de la fille	25	4h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Scripts pour le contrôle du garçon	26	16h	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Scripts pour le contrôle de la fille	27	24h	24	24	16	8	0	0	0	0	0	0	0	0	0	0	0
Gestion des animations de la plante d'attaques à distances	28	4h	4	4	4	4	4	4	0	0	0	0	0	0	0	0	0
Gestion des animations de la plante d'attaques rapprochée	29	4h	4	4	4	4	4	4	0	0	0	0	0	0	0	0	0
Scripts de l'intelligence artificielle de la plante d'attaques à distance	30	8h	8	8	8	8	8	8	8	0	0	0	0	0	0	0	0
Scripts de l'intelligence artificielle de la plante d'attaques rapprochées	31	8h	8	8	8	8	8	8	8	8	0	0	0	0	0	0	0
Création des systèmes de particules et objets pour les pouvoirs de la fille	32	8h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Création des effets pour l'épée et les pas du garçon	33	4h	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Création des particules et effets pour les ennemis	34	4h	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Ajout des effets et systèmes de particules dans les scripts	35	16h	16	16	16	16	16	16	16	16	8	0	0	0	0
---	----	-----	----	----	----	----	----	----	----	----	---	---	---	---	---

« Tableau 3 : deuxième partie du mois »

2. Création des personnages du jeu

2.1. Conception des personnages

Les personnages des deux jumeaux « Umi » et « Almas » ont été imaginés sur le logiciel « Adobe Photoshop Creative Cloud », dessinés à l'aide d'une tablette graphique.

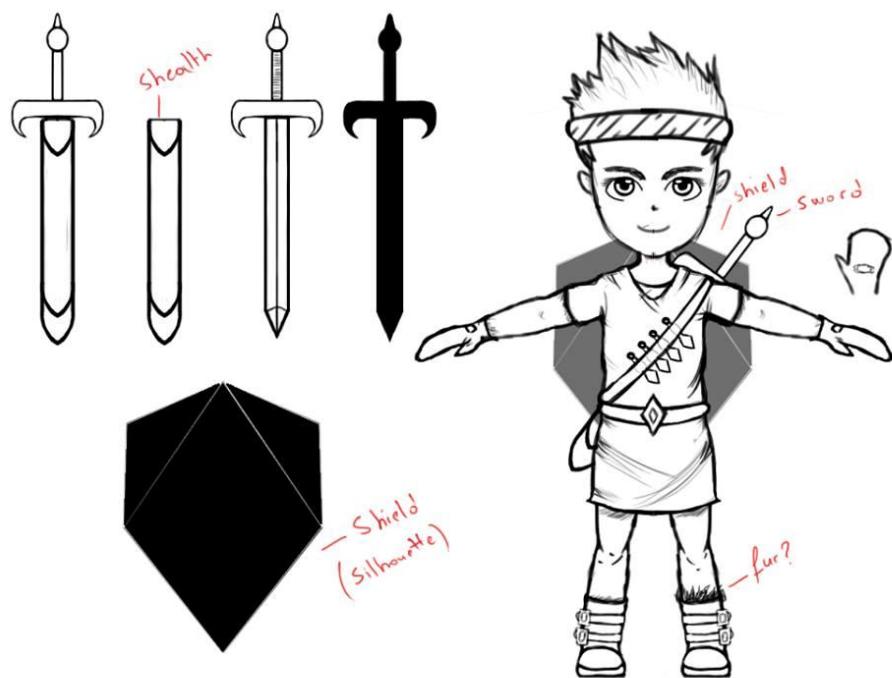
Afin de donner un effet enfantin à nos personnages, nous n'avons pas reproduit les proportions réelles du corps humain mais nous avons plutôt choisi de suivre certaines règles provenant des proportions « **Chibi**⁷¹ ». Proportions inventées par les japonais, les personnages de type « Chibi » ont souvent des têtes disproportionnellement grandes par rapport au corps et certains membres exagérés comme par exemple les mains. Nous avons également reproduit les proportions des yeux du style japonais « Chibi ».

Etant dans un univers fantastique (moyen-âge, forestier et magique), nous avons choisi les habits et armes des personnages de façon à garder une certaine cohérence. Ainsi le garçon possède les armes d'un chevalier (épée, bouclier, couteaux...). Inspiré des légendes du courageux guerrier qui sauve la princesse, « Almas » est le courageux petit frère qui protège sa sœur. Nous avons ajouté des détails au garçon tels que le joyau à sa ceinture ou celui sur le pommeau de son épée pour introduire dans le personnage la notion des joyaux utilisés dans le jeu pour ouvrir certains passages. Par ailleurs, c'est pour cette raison qu'il s'appelle « Almas », le diamant. [voir figure 56]

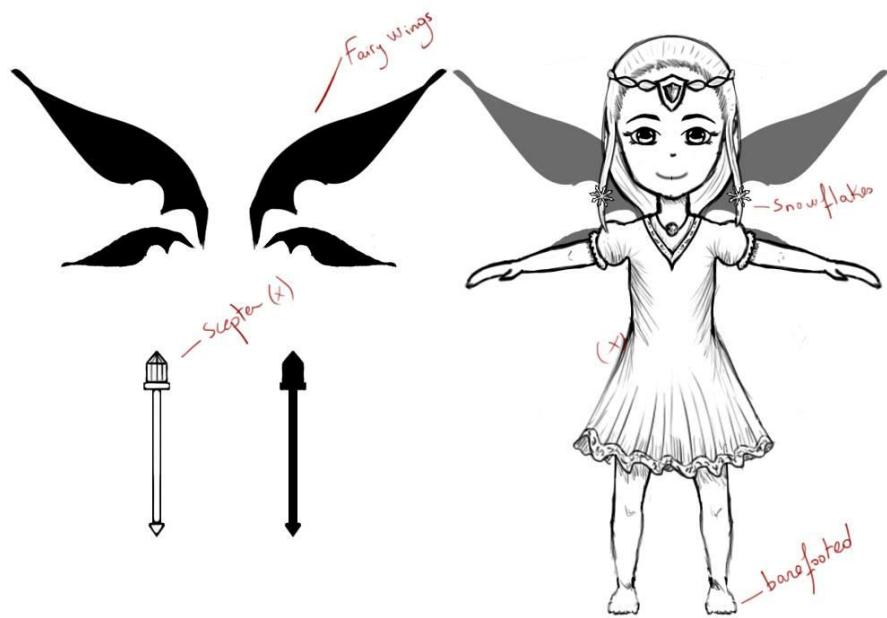
« Umi » quant à elle, est la princesse. « Umi » ne possède pas d'armes, elle se sert de la puissance magique (la glace et les cubes de diamants) pour résoudre des énigmes. Elle préfère réfléchir que se battre et nous l'avons imaginée d'un style vestimentaire épuré, simple mais raffiné. Elle possède les ailes et le sceptre d'une fée ce qui la rend plus éloignée du monde réel contrairement à « Almas ». « Umi » est un rêve, le rêve de voler dans le ciel d'un monde beau et mystérieux. [voir figure 57]

Les ennemis, quant à eux, ont été conçus pour faire partie de l'environnement. Ils sont des plantes et ne contrasteront donc pas avec celui-ci. Basés sur les ennemis de Mario, ils ont été imaginés plus simples que les personnages principaux car il faut que le joueur ne s'attache pas à eux. Ils doivent paraître méchants et amusants en même temps. Ainsi, une touche d'humour a été ajoutée à leur conception (par exemple la grande langue ou l'hélice de feuillage). [voir figure 58]

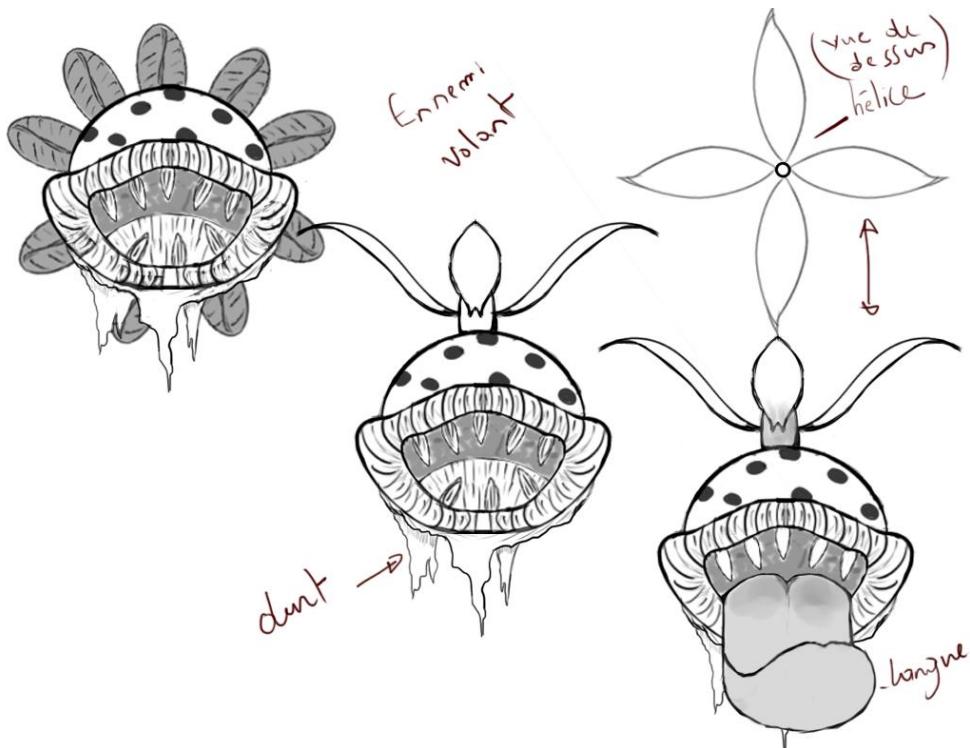
⁷¹ « Mini » en japonais.



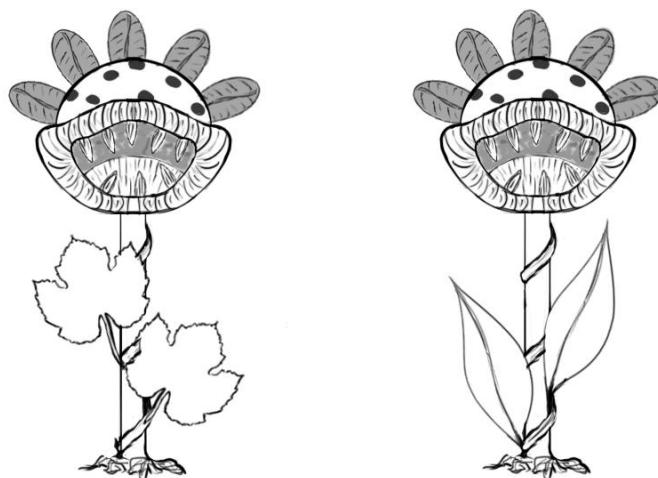
« Figure 56 : Almas »



« Figure 57 : Umi »



« Figure 58 : plante volante »



« Figure 59 : plante terrestre »

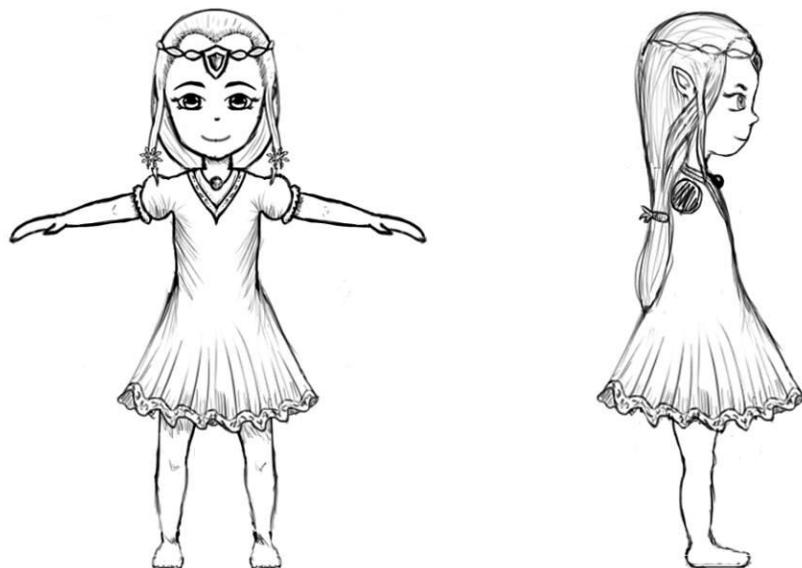
2.2. Création des « Blueprints », modèles pour la modélisation 3D

Une fois une conception validée par l'entreprise, nous pouvions générer les vues de face et de profil qui nous permettront par la suite de modéliser le personnage en 3D.

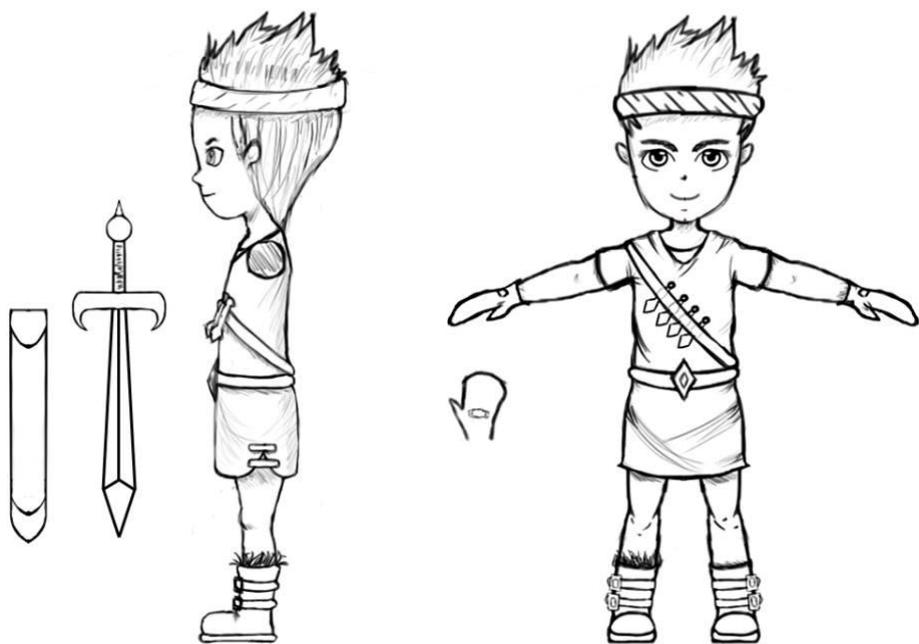
Pour que la modélisation 3D puisse être équilibrée, il faut que les deux vues (face et profil) respectent la même échelle, les mêmes formes et les mêmes proportions [voir figure 60, figure 61, figure

62 et figure 63]. Nous nous servons donc des outils de règles et des repères de Photoshop pour obtenir une cohésion parfaite.

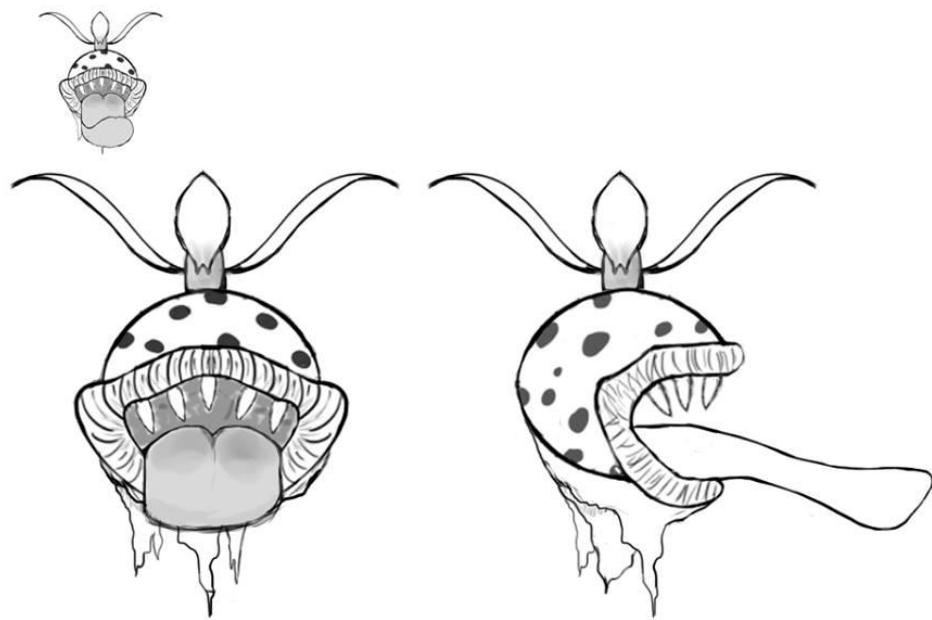
Dans notre jeu, la silhouette et les détails de la vue de profils sont plus importants que ceux de la vue de face car le jeu étant en 2.5D, le personnage est vu de profil durant tout le long de la partie hormis les quelques instants pendant lesquels le personnage change de direction.



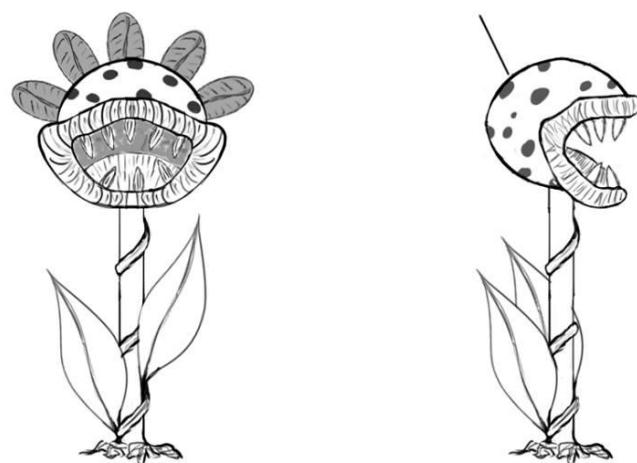
« Figure 60 : blueprint Umi »



« Figure 61 : blueprint Almas »



« Figure 62 : blueprint plante volante »



« Figure 63 : blueprint plante terrestre »

2.3. Modélisation des personnages en utilisant les « blueprints »

Tout comme pour les conceptions, nous avons fait valider les « Blueprints » avant de commencer la modélisation. Dans cette troisième étape, nous nous sommes servis de l'une des méthodes classiques pour faire en sorte que les modélisations soient fidèles aux blueprints et possèdent un flux de polygones optimisé et conforme à l'animation : le « **Box Modeling**⁷² ».

- On crée deux plans 3D de même taille que l'on place perpendiculairement l'un par rapport à l'autre et à distance égale par rapport au centre de la scène 3D,
- On texture les deux plans avec les vues de face et de profil,
- On crée un cube 3D au centre de la scène et on commence à le modifier d'une manière à ce qu'il suit les contours des vues de face et de profil.
- On vérifie la silhouette en réglant la scène 3D de 3DS Max en « **Consistent Colors**⁷³ ».
- Une fois la silhouette totalement fidèle aux blueprints, on corrige les éventuelles erreurs de flux de manière à ne garder des polygones à 4 (ou sinon 3) sommets.

Nous avons également choisi la technique dans laquelle on modélise le corps et les vêtements d'un personnage en un seul « **Mesh** » pour faciliter l'animation. Seuls les accessoires tels que l'épée ou le bouclier sont modélisés en tant que « **Mesh** » distincts.

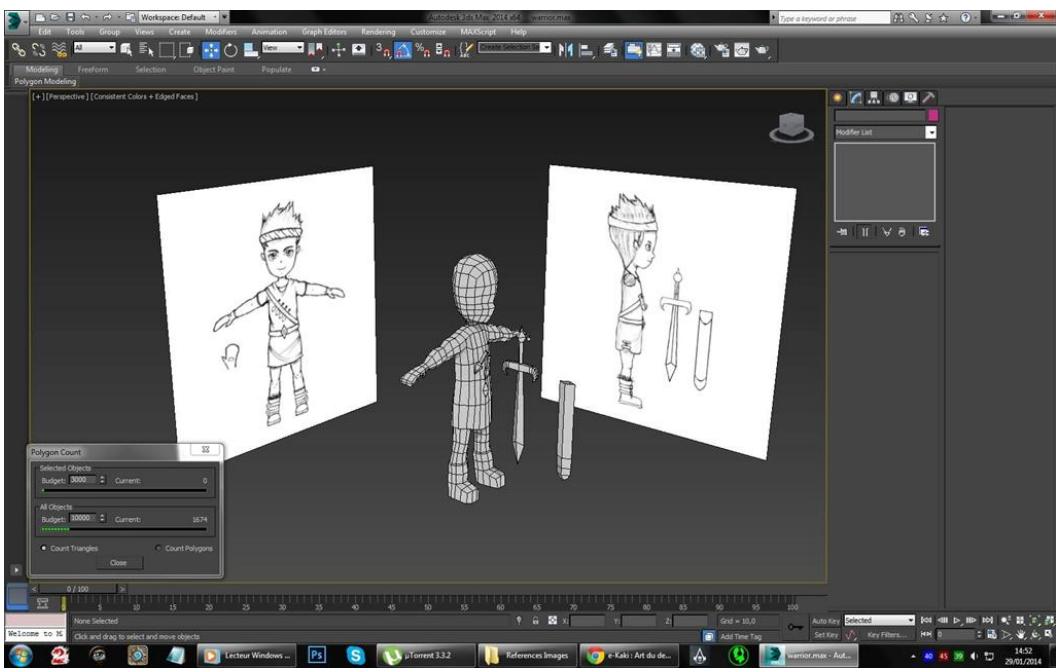
Dans un jeu en 2.5D de type « jeu de plateformes », les personnages sont petits car ils sont vus de loin, et donc le joueur ne peut en apercevoir tous les détails. Ainsi afin de réduire le nombre de polygones utilisés lors de la modélisation, on peut :

- Ne pas modéliser les petits détails tels que les yeux mais plutôt les peindre sur la texture.
- Peindre également les objets plats sur des planes de seulement deux polygones plutôt que de créer réellement l'objet plat (exemple les ailes de la fille).
- Utiliser le compteur de polygones de 3DS Max pour s'imposer à soi-même une limite. Pour être dans la catégorie « **low poly** », il ne faut pas dépasser pour un personnage, une valeur approximative de 3000 polygones [voir figure 64, 65, 66, 67, 68 et 69].

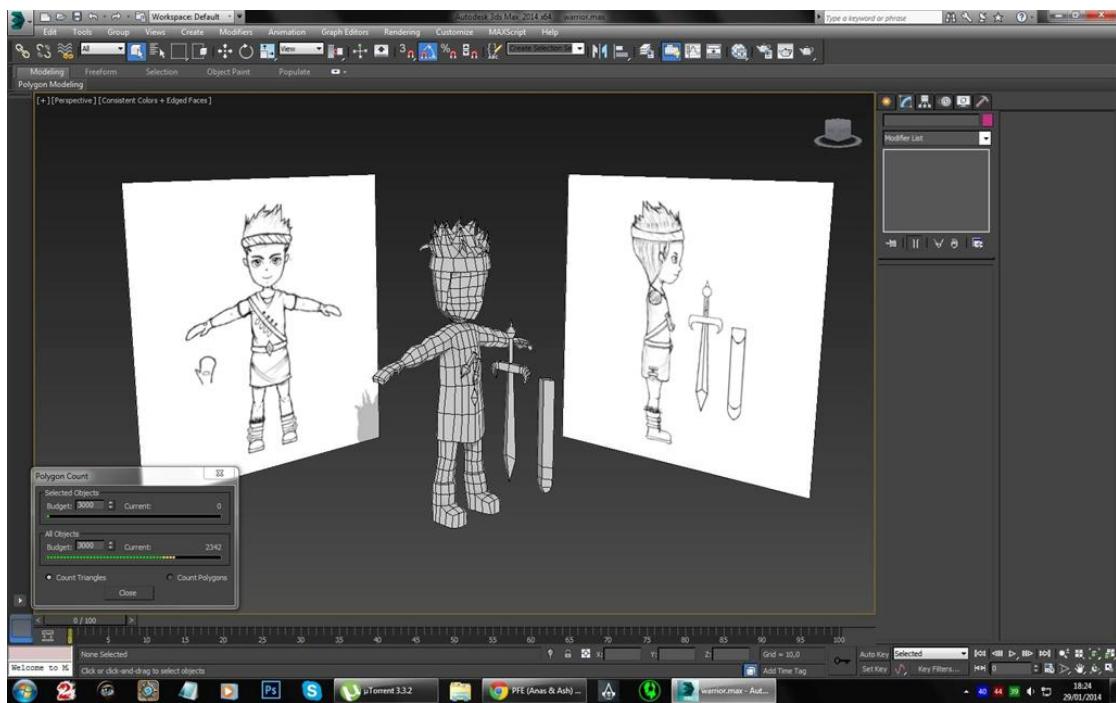
Les personnages étant vus de loin, ses expressions faciales ne seront pas visibles. Pour leur créer un charme et un effet comique, nous avons été obligés de compter sur leurs formes corporelles. Ainsi, nos personnages sont assez « arrondis » et « petits » et nous avons modélisé de façon à ce que les polygones créent cet effet.

⁷² La technique utilisée pour modéliser à partir d'un cube.

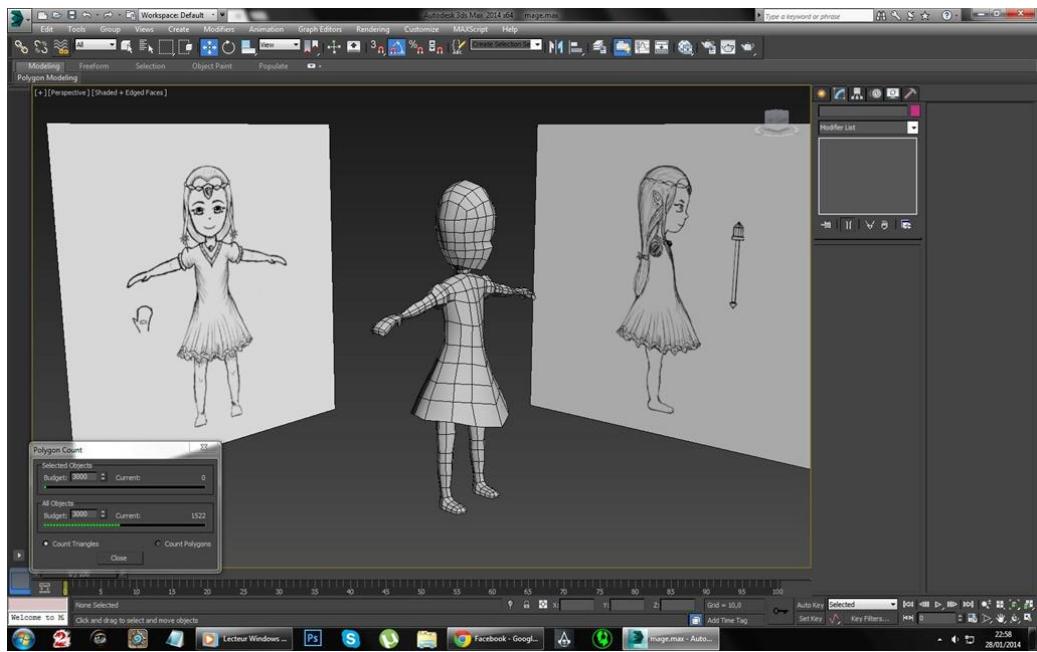
⁷³ Affichage en couleurs pures (sans ombrages ni lumière).



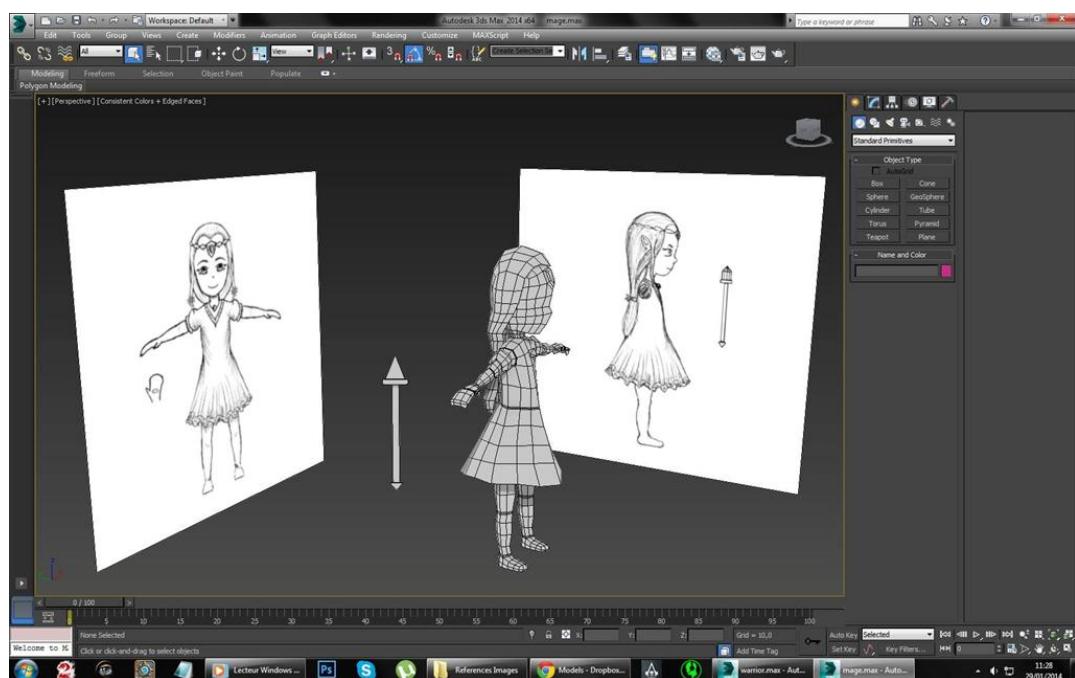
« Figure 64 : modélisation low poly – Almas »



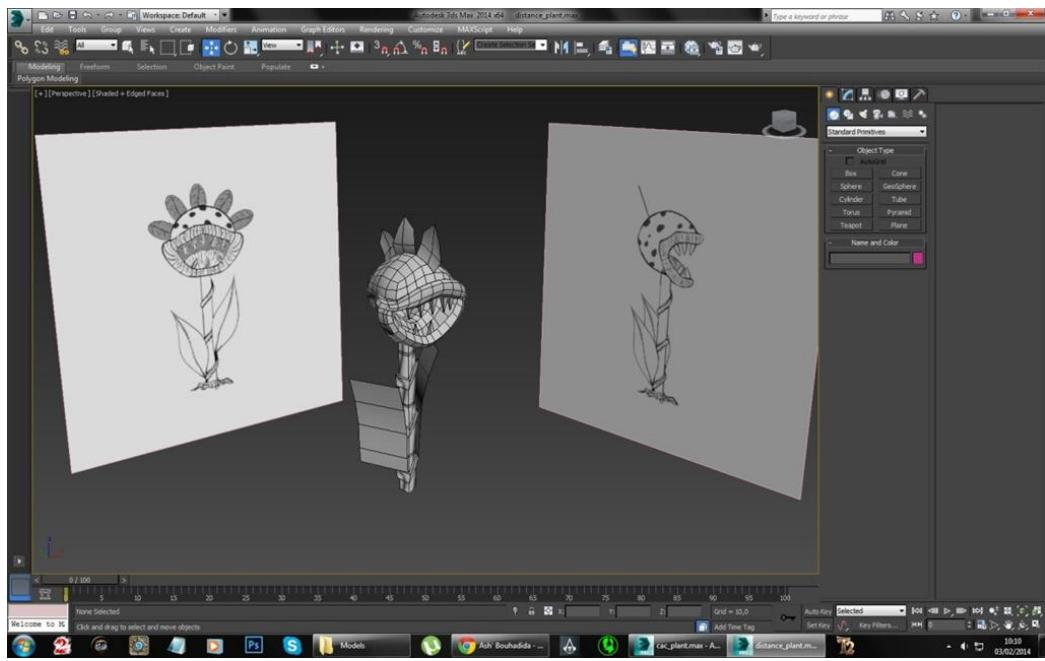
« Figure 65 : limite de polygones – Almas »



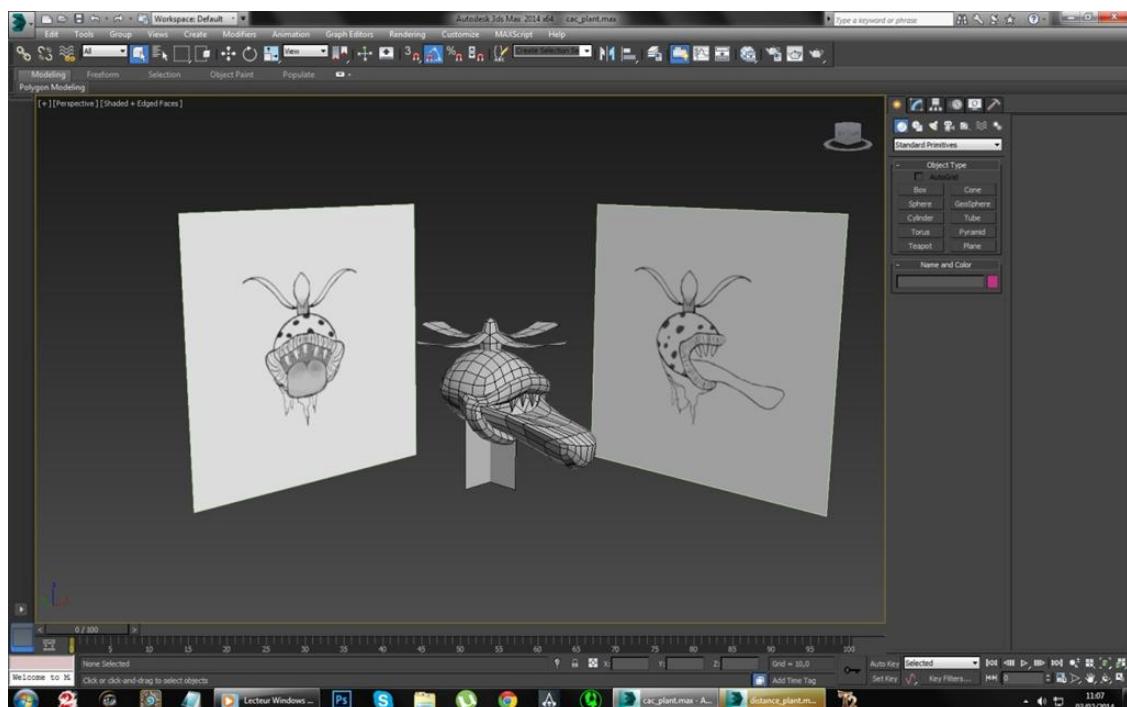
« Figure 66 : modélisation low poly – Umi »



« Figure 67 : limite de polygones – Umi »



« Figure 68 : modélisation low poly - plante volante »



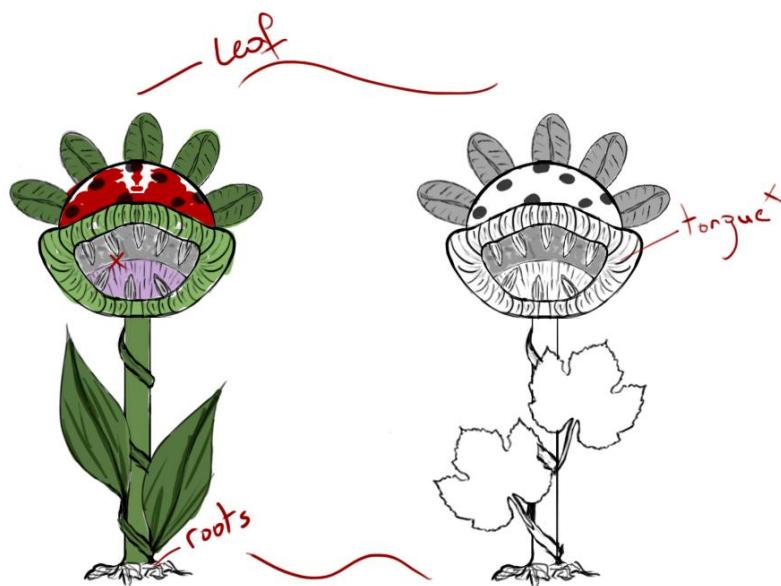
« Figure 69 : modélisation low poly - plante terrestre »

2.4. Texture et coloration des personnages

Le charme d'un personnage n'est complet qu'une fois qu'il est coloré. Il est nécessaire que les couleurs des personnages soient en accord les uns avec autres mais aussi avec l'environnement. Il faut également que les couleurs évoquent un sentiment.

2.4.1. Les ennemis

Nous avons choisi le rouge pour les plantes. Le rouge fait peur et attire le regard, il est très voyant, choquant et évoquant le danger, la colère ou le feu. Pour ces raisons, les personnages ennemis sont souvent rouges et noirs dans les jeux vidéo. Nos ennemis étant dans plantes, il a fallu également ajouter une touche naturelle et nous avions alors le choix entre le marron et le vert. Nous avons préféré le vert car plus éloigné du rouge que le marron, il permettait d'ajouter un contraste. Enfin, pour signaler le fait que la plante ne doit pas toucher la fille car sa langue est empoisonnée, nous avons choisi de colorer la langue en violet, couleur souvent utilisée pour représenter le poison et autres substances toxiques [voir figure 70, 71, 72, 73 et 74].



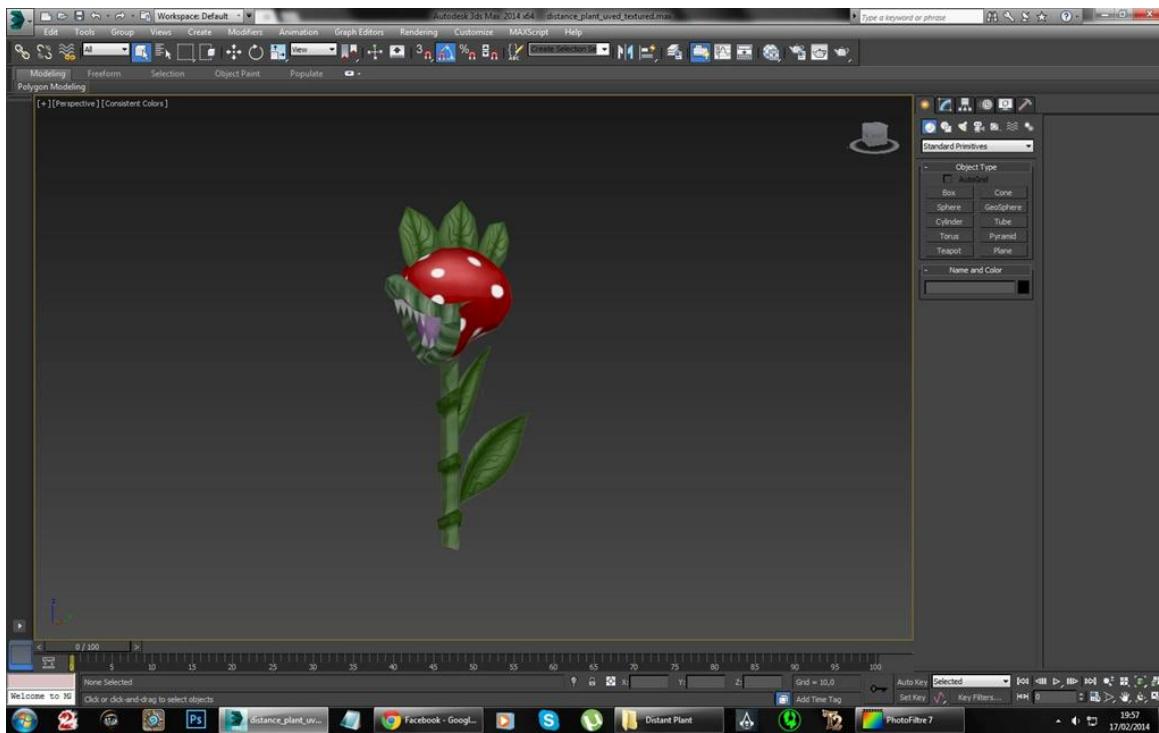
« Figure 70 : choix de couleurs - plante terrestre »



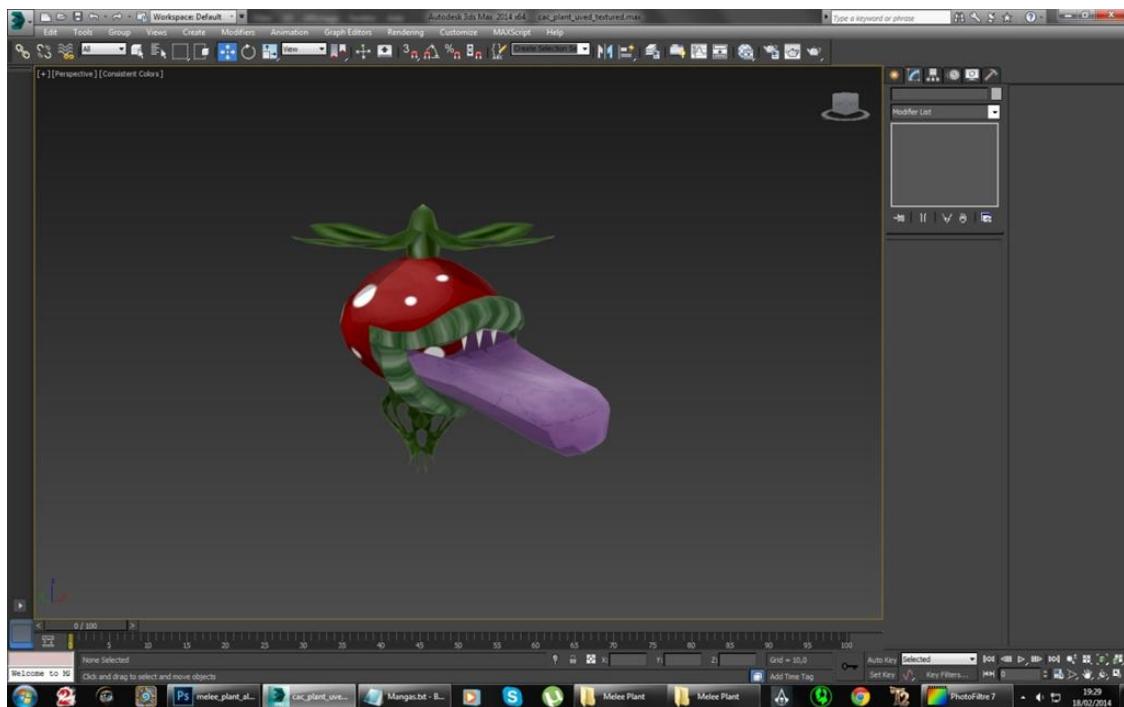
« Figure 71 : UV peints - plante terrestre »



« Figure 72 : UV peints - plante volante »



« Figure 73 : modélisation - plante terrestre »



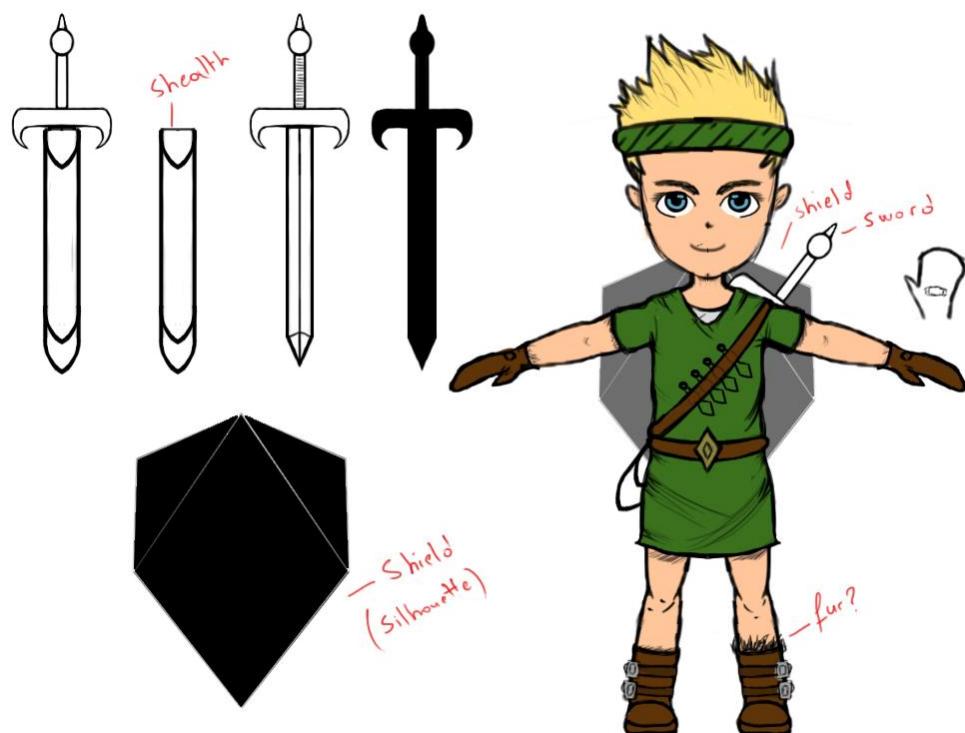
« Figure 74 : modélisation - plante volante »

2.4.2. « Almas »

« Almas » possède deux types de coloration : ses caractéristiques corporelles et ses habits. Pour les habits, nous avons choisi les couleurs de la nature : le vert et le marron. Nous avons ensuite décidé qu'ils seraient, lui et sa sœur, blonds avec des yeux bleus. Ces caractéristiques sont souvent, dans la littérature notamment, celles des elfes vivant dans les bois (comme par exemple dans « The Lord Of The Rings ») [voir figure 75]. Nos personnages étant magiques et vivant au milieu d'une forêt, cela correspondait tout à fait [voir figure 76, 77 et 78].



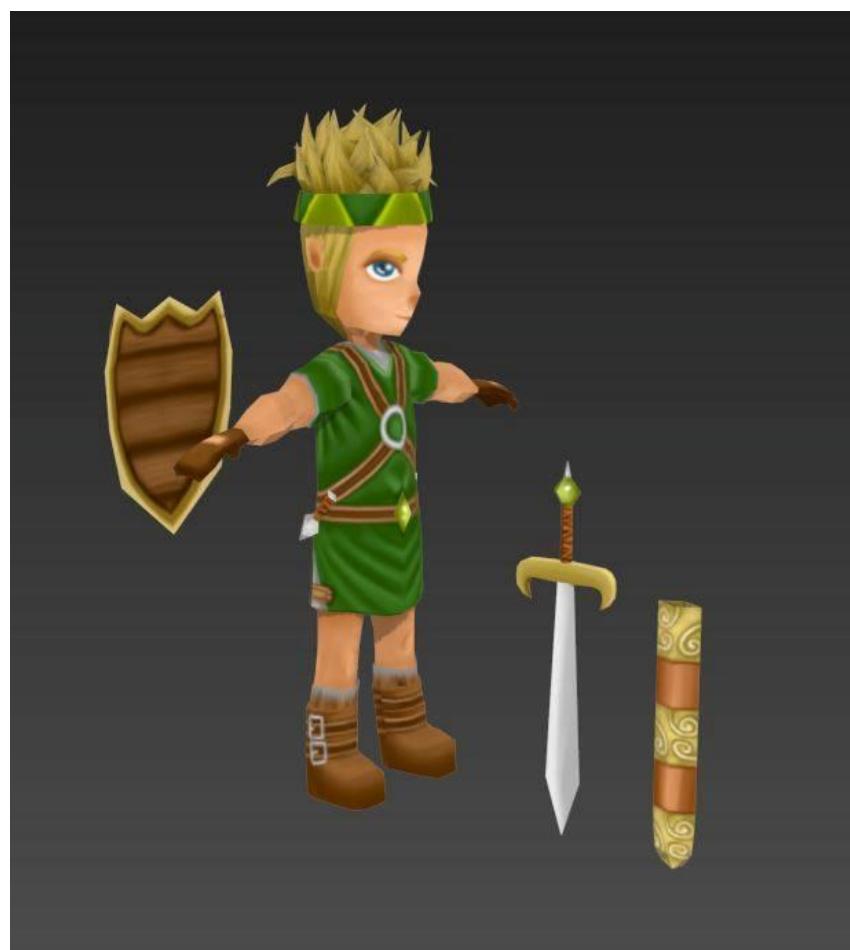
« Figure 75 : Un elfe de « Le Seigneur des Anneaux » »



« Figure 76 : choix de couleurs - Almas »



« Figure 77 : UV peints – Almas »

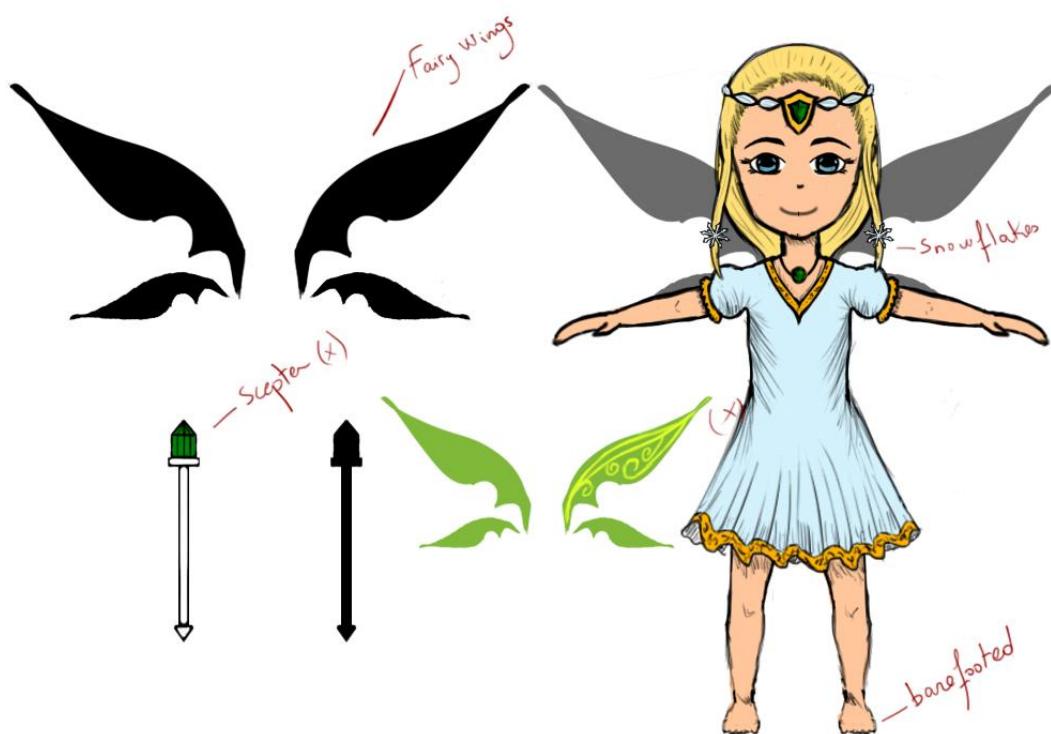


« Figure 78 : modélisation finale – Almas »

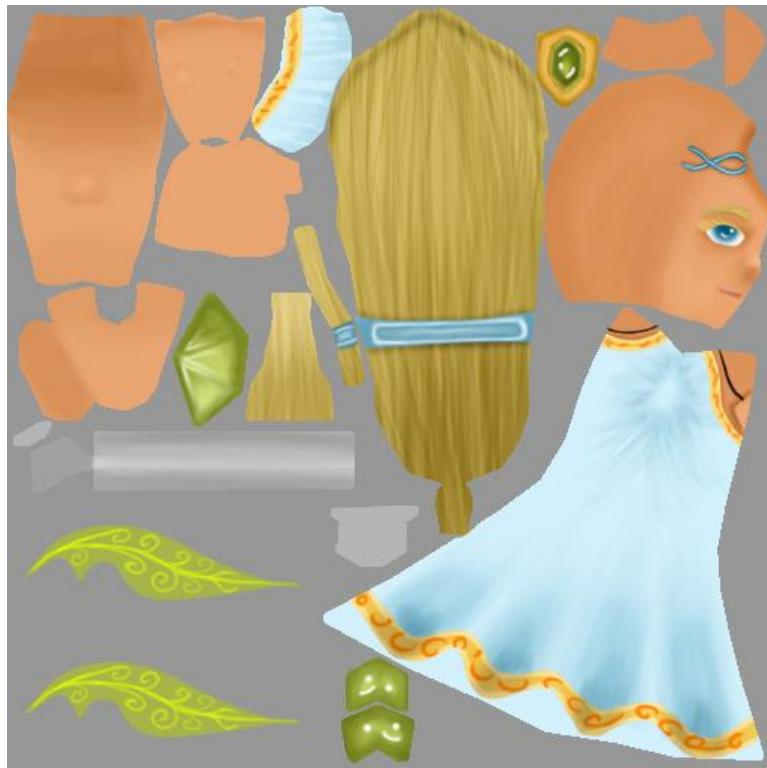
Quant aux armes, elles possèdent la couleur dorée pour évoquer la noblesse et le courage du personnage. Les autres couleurs sont très habituelles pour ce type d'armes : une lame blanche et un manche en bois.

2.4.3. « Umi »

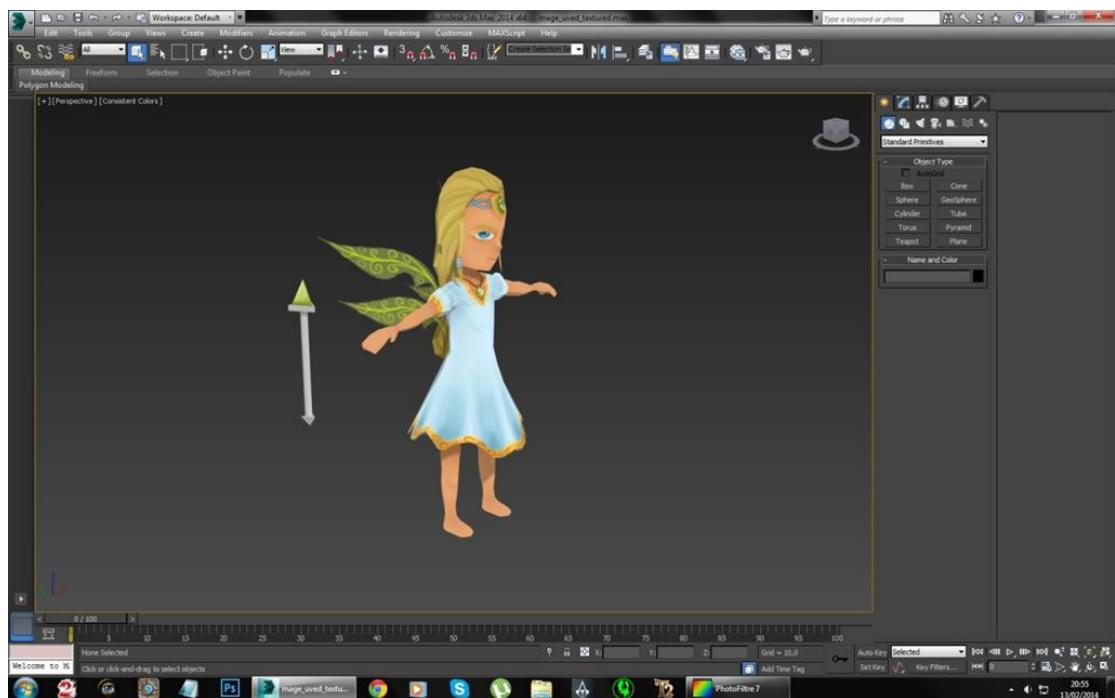
« Umi » étant jumelle du garçon, possède les mêmes caractéristiques corporelles. Cependant, sa personnalité change. Les habits qu'elle porte révèlent plutôt cette personnalité. Ainsi, elle porte des habits clairs d'un bleu presque blanc. Cela inspire la pureté. Elle porte également du bleu pour qu'il y ait un rappel de la couleur des yeux. La magie qu'elle utilise étant liée à la glace, le blanc et le bleu de ses yeux et de ses vêtements créent un fort lien entre la couleur et la nature de sa magie. Son sceptre est gris, en métal, car le métal et le gris évoquent le froid et la sobriété qui sont très présents dans le personnage. Enfin, pour que la fille possède tout de même un lien avec l'environnement, nous avons coloré ses ailes en vert [voir figure 79, 80 et 81].



« Figure 79 : choix de couleurs – Umi »



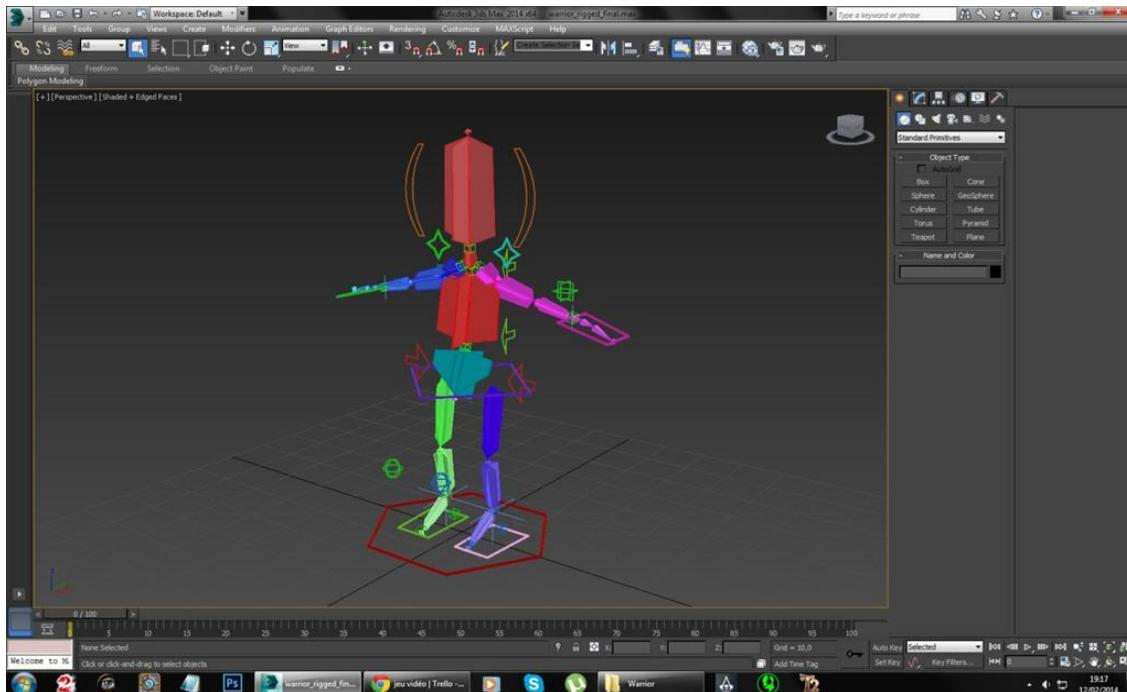
« Figure 80 : UV paints – Umi »



« Figure 81 : modélisation finale – Umi »

3. Animation des personnages

Pour l'animation des personnages, nous avons utilisé la méthode classique d'animation des jeux vidéo. Nous avons donc créé un « **rig** » simple mais suffisant pour les animations dont nous avions besoin à l'aide des outils 3DS Max : les « **bones** », les « **controllers** » ou encore les « **IK solvers** ». Après cela, nous avons lié le squelette au corps, le « **skinning** ». [voir figure 82]



« Figure 82 : rigging des personnages »

Nous avons choisi de mettre toutes les animations d'un personnage sur un seul fichier. Par exemple, sur le fichier de « Almas », on peut trouver :

- « **Idle**⁷⁴ », animation d'inactivité qui peut se lire en boucle sans coupure. Dans « Idle », le personnage ne bouge presque pas, il se contente de remuer légèrement et il est en attente.
- « **Run**⁷⁵ », une animation de course également en boucle.
- « **Attack** », une animation de frappe avec l'épée qui ne boucle pas mais finit avec un retour vers la position de début de « Idle ».
- « **Protection** », une animation de protection avec le bouclier.
- « **Stop protection** », une animation d'arrêt de protection qui commence avec la position finale de « Protection » et finit avec la position initiale de « Idle ».
- « **Die**⁷⁶ », l'animation de mort du personnage qui commence avec la position finale de « Idle » et finit avec le personnage allongé sur le sol.

⁷⁴ Animation d'inactivité.

⁷⁵ Animation de course.

⁷⁶ Animation de mort.

4. Création des pouvoirs

Il nous a fallu également créer graphiquement les effets de pouvoirs magiques et des actions qui seront présents dans le jeu des personnages [voir figure 83]. Par exemple, dans les jeux, si un personnage reçoit un coup ou une attaque, un effet 2D s'applique autour du personnage pour indiquer le coup.



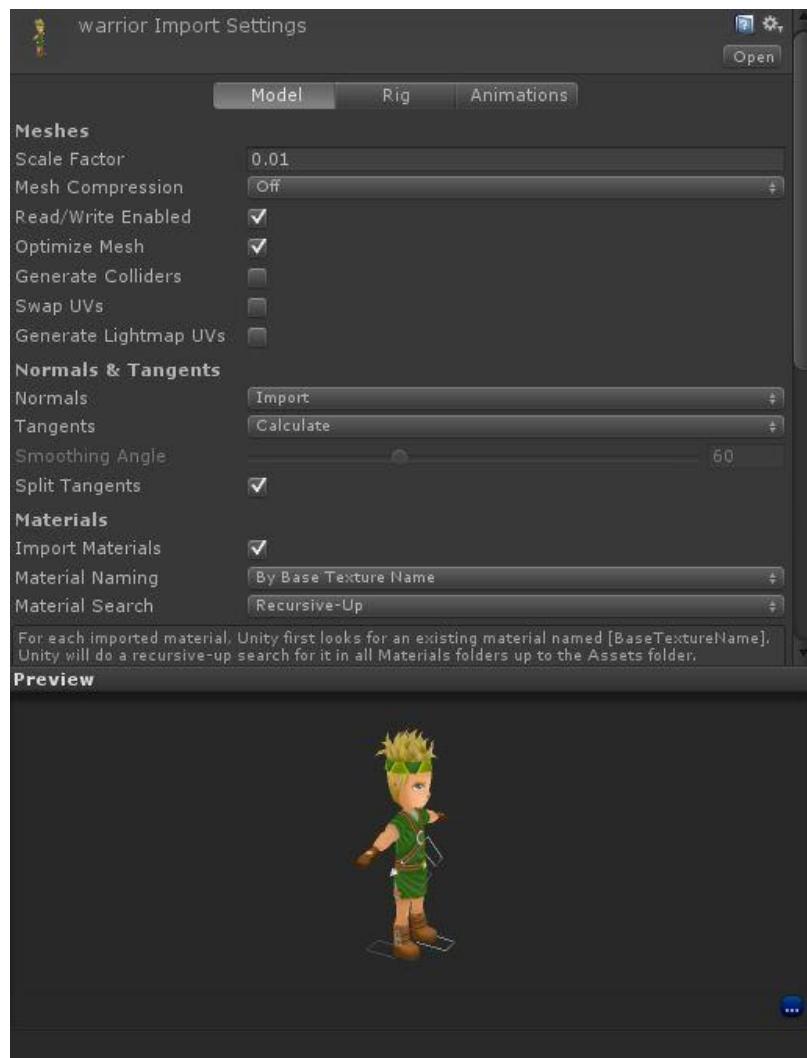
« Figure 83 : effets 2D »

5. Intégration des personnages

5.1. Intégration des personnages dans Unity

Avant de commencer à programmer les personnages, il a fallu effectuer plusieurs réglages dans Untiy pour pouvoir les utiliser :

- Importer le modèle au format général « **FBX⁷⁷** » compatible et qui peut comporter les animations. [voir figure 84]



« Figure 84 : importation des .fbx »

- Importer la texture au format « **TGA⁷⁸** » qui à l'avantage de gérer la « couche alpha » pour les zones de transparence et qui n'effectue pas de compression ou de perte de qualité.
- Créer un « **shader** » à partir de ceux de Unity qui peut gérer la coloration des deux côtés d'un polygone, cette fonctionnalité s'appela le « **cull off** ». Dans Untiy, les matériaux n'affichent qu'un côté des polygones, de l'autre, ils sont invisibles. Nos personnages possèdent des éléments de type « **plane** » comme les ailes de la fille ou les feuilles des plantes qui doivent être vues des deux côté et nous avons donc du modifier la programmation des « **shader** » de

⁷⁷ Il est utilisé pour assurer l'interopérabilité entre les applications de création contenu numérique.

⁷⁸ « Targa ».

Untiy en ajoutant une simple ligne. [voir figure 88]

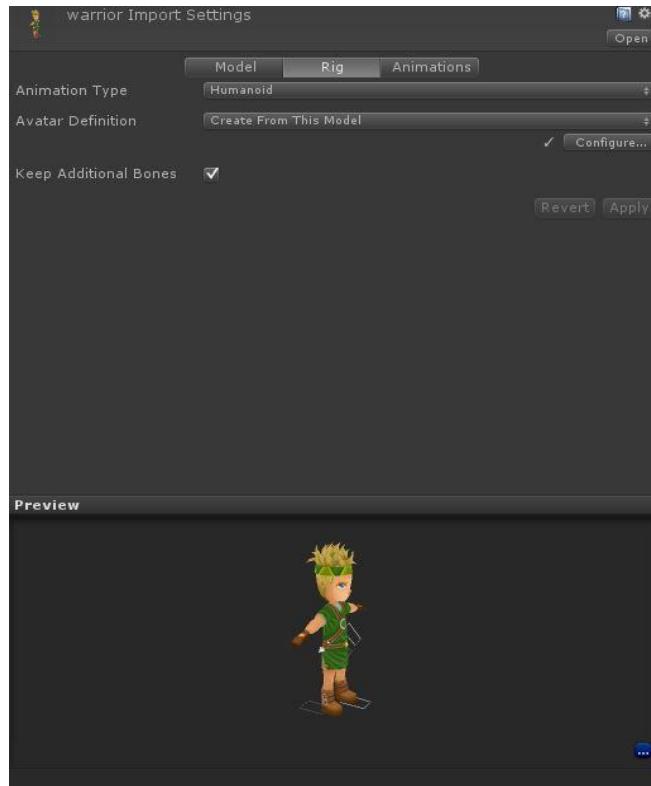
```
Shader "theTniws/theTwinsDiffuse" {
Properties {
    _Color ("Main Color", Color) = (1,1,1,1)
    _MainTex ("Base (RGB)", 2D) = "white" {}
}
SubShader {
    Tags { "RenderType"="Opaque" }
    LOD 200
    Cull off

    Pass {
        Name "FORWARD"
        Tags { "LightMode" = "ForwardBase" }

    
```

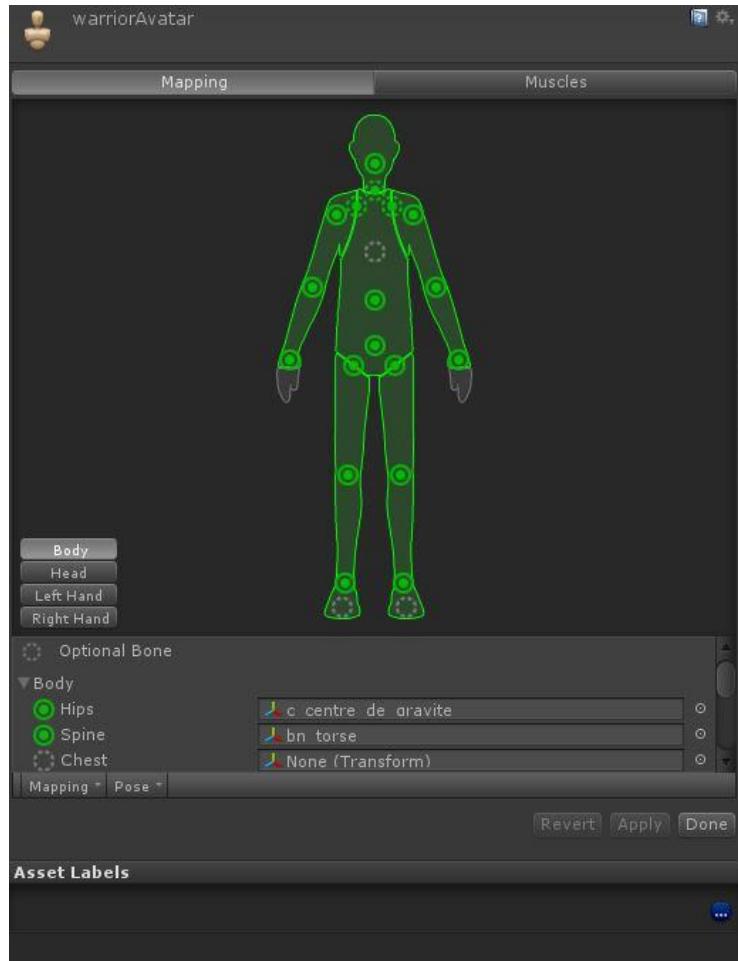
« Figure 85 : modification de shader »

- Choisir le type d'animation convenant pour des animations avec « **Rigging** ». Le type qui correspond s'appelle « **Humanoid** » [voir figure 86].



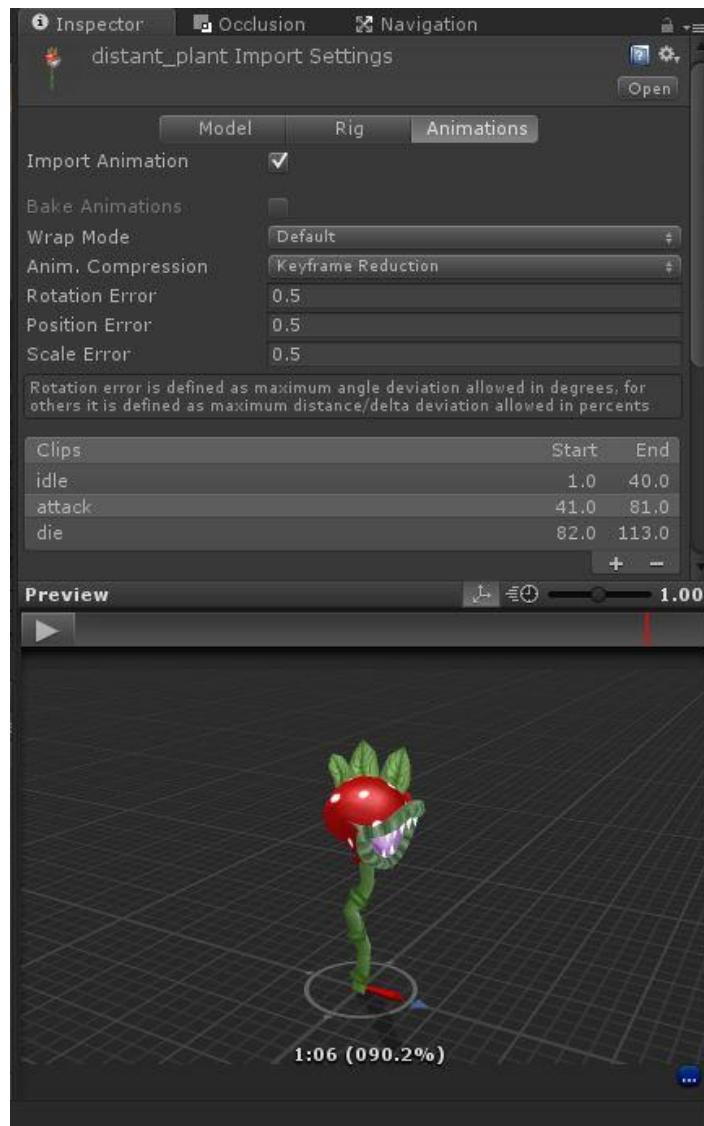
« Figure 86 : type du rig »

- Après cela, il faut faire correspondre le gabarit de squelette de Unity avec celui que l'on a créé dans 3DS Max [voir figure 87].



« Figure 87 : affectation de bones »

- Toutes les animations étant enregistrées sur un seul fichier, elles sont importées comme une seule animation. Il nous a donc fallu la découper dans Unity en fonction des « frames ». Par exemple, l'animation « Idle » du garçon débute à la position 0 et finit à la « frame » 24. Pour chaque animation créée, il faut spécifier si elle doit déplacer le personnage ou encore si elle doit être lue en boucle [voir figure 88].



« Figure 88 : fichier d'animations »

- Enfin, il faut supprimer tous les « **components**⁷⁹ » inutiles ajoutés par Unity. Par exemple le composant « **Animation** » est créé automatiquement et doit être remplacé par « **Animator** », plus adapté pour les animations utilisant un squelette.

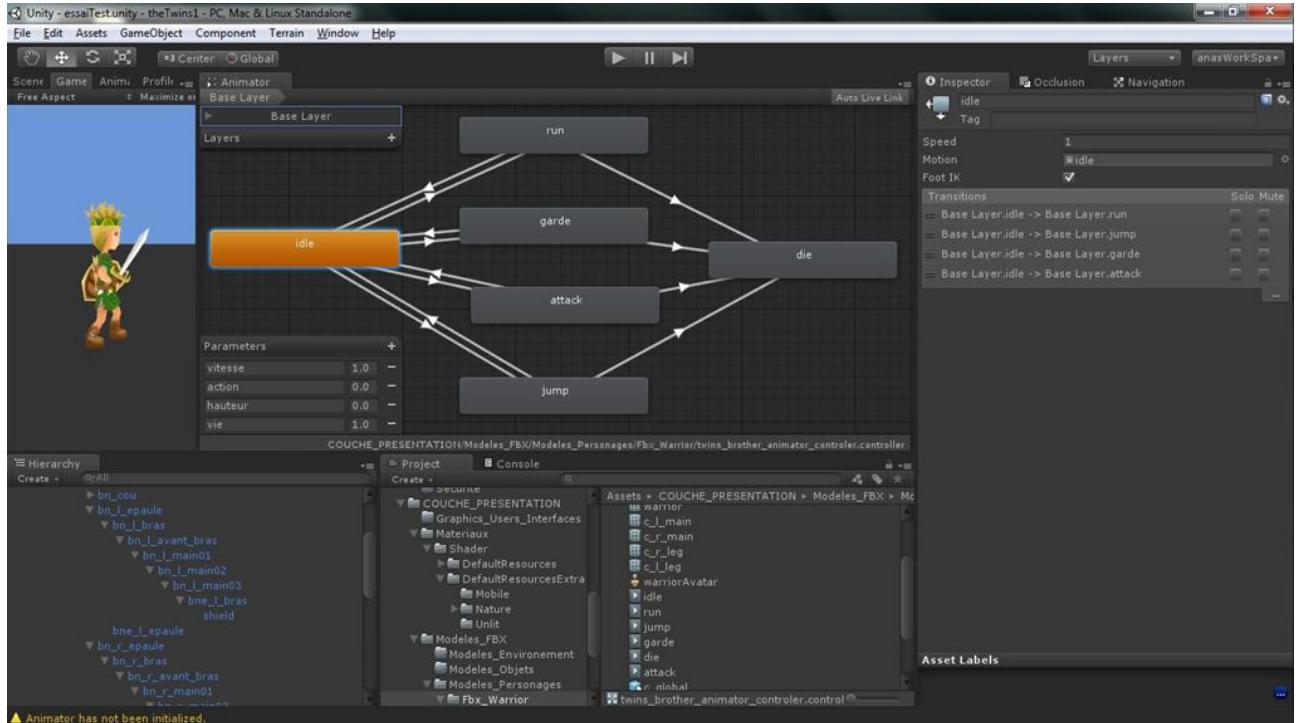
5.1. Gestion des animations avec Mecanim

L'un des outils les plus puissants de Unity est « **Mecanim** ». Mecanim fonctionne en créant un arbre ou graphe d'animation [voir figure 89] selon les principes suivants :

- On place toutes les animations d'un personnage en tant que **nœud**. On choisit « Idle » comme animation initiale. Pour chaque animation, il faut régler la vitesse.
- On crée des variables représentant les actions du personnage par exemple « Action », « Vitesse » ou encore « Hauteur » qui pourraient le faire changer d'animation et donc aller d'un nœud à un autre.
- On crée des branches orientées entre les nœuds. Par exemple, si un personnage peut passer de l'animation « Idle » à l'animation « Attack » quand on appuie avec le bouton gauche de la souris. Il faut créer une branche entre le nœud « Idle » et le nœud « Attack ».

⁷⁹ Un composant.

- Il faut également préciser pour chaque branche quelle est la variable qui déclenche le passage d'une animation à une autre. Pour reprendre notre exemple, le passage de « Idle » à « Attack » en utilisant la branche se fait uniquement lorsque la variable « Action » prend la valeur « 1 ».



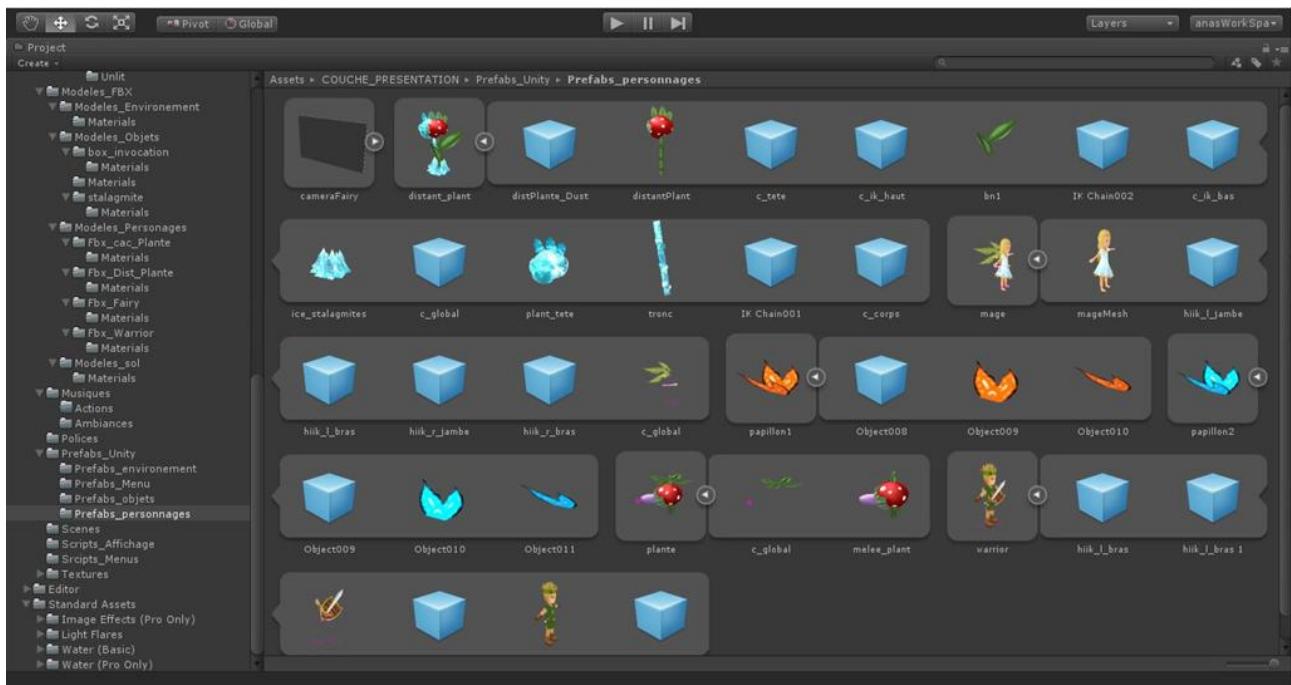
« Figure 89 : arbre Mecanim »

- Enfin, pour chaque personnage, il faut créer un script qui modifiera les valeurs des variables de Mecanim en fonction des touches appuyées avec le clavier par exemple. Le passage entre deux animations n'est pas coupé parce que Mecanim crée des fusions lors du passage entre elles.

5.1. Création du prefab Unity

Une fois que l'on a fini le script du personnage, les réglages, l'arbre Mecanim, la mise en place des matériaux et les autres ajouts tels que des « components » que l'on utilise sur le personnage, on l'enregistre en tant que « **Prefab**⁸⁰ » [voir figure 90]. Un « prefab » est un modèle prêt à être copié et réutilisé plusieurs fois. Ainsi, chaque objet que l'on va instancier sur la scène (comme les personnages, les attaques ou autres objets ne faisant pas partie du décor) doit être enregistré en tant que «prefab».

⁸⁰ Élément préfabriqué.



« Figure 90 : une liste de prefabs »

6. Programmation des personnages

6.1. Introduction à la Scripting Reference

Le moteur Unity possède deux éléments internes d'apprentissage du fonctionnement du logiciel. Le manuel interne, utile pour apprendre toutes les fonctionnalités de Unity et la « **Scripting Reference** ». Celle-ci est disponible à partir du logiciel Unity ou sur internet.

(<https://docs.unity3d.com/Documentation/ScriptReference>)

Les quatre principes de base sont :

- Chaque « **script** » existant dans Unity est une classe.
- Toutes les classes héritent de la classe « **Object** ».
- Un **gameObject** auquel on a ajouté deux classes, possède en fait deux instances de ces classes dans une variable de type liste de « **components** ».
- Toutes les classes créées par un utilisateur qui souhaite les « appliquer » à un objet 3D doivent hériter directement ou indirectement de la classe « **MonoBehaviour** ». L'objet 3D héritera donc de cette classe en plus de celles dont il hérite déjà. Ces classes deviennent donc des « **Components** ».

Ainsi, on peut comprendre que les classes déjà présentes dans Unity forment une hiérarchie dont la racine est « **Object** ». On comprend également que chaque fois qu'un utilisateur crée un script dans son projet, il crée une nouvelle classe dans la hiérarchie et donc une « feuille de l'arbre » en dessous de « **MonoBehaviour** » ou de la classe dont elle hérite.

Voici une liste réduite des classes les plus utilisées de Unity. Toutes ces classes héritent de « **GameObject** » :

- « **Transform** » : tout objet de Unity visible dans la hiérarchie du projet doit posséder une instance de cette classe. Elle contient en variable la position de l'objet sur trois axes, sa taille sur trois axes et sa rotation sur trois axes. Elle contient également son « **name** », son « **tag** » et son « **layer** ». Enfin, cette classe contient la hiérarchie de l'objet et permet donc d'accéder aux sous-objets inclus dans celui-ci ou encore celui qui l'inclut.
- « **MeshFilter** » : cette classe contient le maillage 3D importé en .FBX.
- « **SkinnedMeshRenderer** » : cette classe hérite de « **Rendrer** » comme « **MeshRenderer** » mais à la différence de cette dernière, elle n'affiche pas les objets simplement, elle les affiche à l'écran (via le moteur de rendu) en tenant compte du squelette.
- « **Animation** » et « **Animator** » : Ce sont deux « components » permettant à l'objet 3D d'être animé avec la différence que « **Animator** » tient compte de « **Mecanim** ».
- « **Rigidbody** » : l'ajout d'une instance de cette classe permet de spécifier au moteur physique « **PhysX** » de NVIDIA qu'il doit tenir compte de la physique. Si c'est le cas, l'objet possède alors une masse et d'autres attributs liés à la gravité et aux forces.
- « **BoxCollider** » : cette classe crée autour de l'objet une forme invisible qui permet de détecter les collisions entre les objets. « **BoxCollider** » crée la forme la plus simple, un cube. Ainsi, c'est celui que nous avons utilisé car il est très optimisé pour la mémoire. « **MeshCollider** » est le plus réaliste visuellement car la forme de collision est exactement la même que celle du « **MeshFilter** », cependant elle est inutilisable en grand nombre car elle nécessite énormément de calcul.

Il est fondamental d'avoir ces notions pour réellement maîtriser la puissance du logiciel.

6.2. Structure de nos classes et procédure générale

Pour faciliter la lecture et la maintenance future de notre projet, nous avons créé un modèle de structure pour nos classes. Ce modèle se sert de la technique des commentaires pour diviser le code en blocs. Chaque bloc est placé dans un ordre constant de lecture. Voici le modèle réutilisable résumé dans la figure ci-dessous :

```
/* Ici une description de la classe */  
class Nom_de_la_classe extends MonoBehaviour  
{  
//_____  
// VARIABLES DE CLASSES  
//_____
```

Dans cette zones, les variables privée et publiques

```
//_____  
// GETTERS ET SETTEURS  
//_____
```

Ici les getteurs et setteurs des variables privées si il y en a

```
//  
// METHODES HERITEES DE MONOBEHAVIOUR  
//
```

Dans cette zone, les méthodes de MonoBehaviour telles que « Update » ou encore « Start »

```
//  
// AUTRES METHODES  
//
```

Dans cette zone, les nouvelles méthodes codées par nous même.

```
//  
//  
}
```

« Figure 91 : structure de scripts »

Il faut également noter que :

- Les variables publiques sont nécessaires si elles doivent prendre pour valeur un « prefab » ou un autre objet du projet (texture, matériaux, musique...) et que cette valeur n'est pas disponible dans la hiérarchie. Dans le cas contraire, et donc l'utilisation de variables privées avec « getter » et « setter » est possible, nous avons préféré utiliser les variables privées. Une variable publique constitue une faille, bien que très légère, dans le code mais surtout nécessite plus de calcul qu'une variable privée.
- Si l'un de nos scripts accède à un autre, l'autre script est enregistré dans l'une des variables privées au début du code. En effet, accéder à un « component » nécessite un grand temps de calcul, l'accès à une case mémoire d'une variable beaucoup moins. Nous procédons toujours de la même manière :

```
private var _NomDuScript : NomDuScript = this.gameObject.GetComponent(NomDuScript) ;
```

Par exemple :

```
private var _transform :Transform = this.gameObject.GetComponent(Transform) ;
```

« Figure 92 : déclaration de variables optimisée »

- Pour l'étude du travail et la future maintenance, chaque méthode utilisée et chaque attribut créé est commenté d'une description rapide. Ainsi même un lecteur ne connaissant pas le langage arriverait à en discerner le sens.

6.3. La classe de contrôle du personnage « Almas »

Le personnage Almas avec lequel on joue dans le jeu est une classe appelée « **warrior** » elle correspond au « **prefab** » nommé « **warrior**⁸¹ ». Le « **warrior** » possède un « **BoxCollider** », un « **Rigidbody** », un « **Transform** », des « **components** » liés au jeu en multi-joueurs que nous détaillerons par la suite et trois scripts que nous avons créé : « **WarriorInit** », « **DustControl** » et « **WarriorControl** ». Le prefab contient également dans son « **Transform** » un sous objet « **warrior** » qui possède le « **SkinnedMeshRenderer** », l'**« Animator** » et le « **MeshFilter** ».

Nous allons détailler les deux scripts que nous avons créés.

6.3.1. WarriorInit

Le script est assez simple, il vérifie à l'instanciation du personnage sur la scène (via la fonction « **Start()** » de « **MonoBehaviour** ») que seul le joueur qui l'a créé peut exécuter le script « **WarriorControl** » et donc contrôler le personnage. Le script vérifie aussi que seule la « **caméra** » (**gameObject** héritant de la classe « **Camera** » et qui peut afficher la scène à l'écran) du joueur qui contrôle Almas exécute le script « **CameraFrere** » de suivi d'Almas.

6.3.2. WarriorControl

Ce script est assez long et assez complexe. Il a quatre principales utilités :

- La fonction « **Clavier()** » exécutée via la fonction « **FixedUpdate()** » donc entre 24 et 60 fois par seconde. Cette fonction change les variables de **Mecanim** en fonction des touches du clavier appuyées. Ainsi, cette fonction change les animations jouées par le personnage. Cette fonction change les valeurs des variables du « **component Transform** ». Ainsi, elle fait tourner ou avancer le personnage. Les mouvements verticaux se font eux en utilisant le « component **Rigidbody** ». Normalement, on essaie de travailler avec l'un ou l'autre et non pas les deux ensembles mais avec les contraintes de position dues à la 2.5D, il nous a fallu réussir à les combiner. Le personnage, et tous les autres d'ailleurs, possèdent la valeur zéro pour leurs positions sur l'axe « **x** ». Ils sont donc tous sur une même bande qui est le chemin de déplacement [voir figure 93 et figure 94].



« Figure 93 : déplacements »

⁸¹ « Guerrier »



« Figure 94 : saut »

- La fonction **Control()** qui, exécutée aussi souvent que **Clavier()**, vérifie que si le personnage tourne, il tourne en progression et non tout d'un coup. Elle vérifie aussi que si le personnage n'a plus de vie, il joue l'animation « **Die** » et revient tout au début du jeu.
- Les fonctions de détection de collisions « **OnCollisionEnter** » (avec également **Stay** et **Exit**) et « **OnTriggerEnter** ». La différence entre « **OnCollisionEnter** » est la nature de l'élément de collision, dans un cas c'est un objet que l'on ne peut traverser et dans l'autre un objet dont la collision n'est pas visible. Elles vérifient toutes les deux si le personnage est bien au sol, et que donc il peut sauter mais aussi que s'il touche un objet mortel (poison, flèche...) sa vie diminue.
- Les petites fonctions d'actions envoyées en réseaux. Par exemple, une fonction qui active une traînée lumineuse (crée par la classe « **TrailRenderer** ») derrière l'épée lorsque le personnage donne un coup d'épée [voir figure 95].

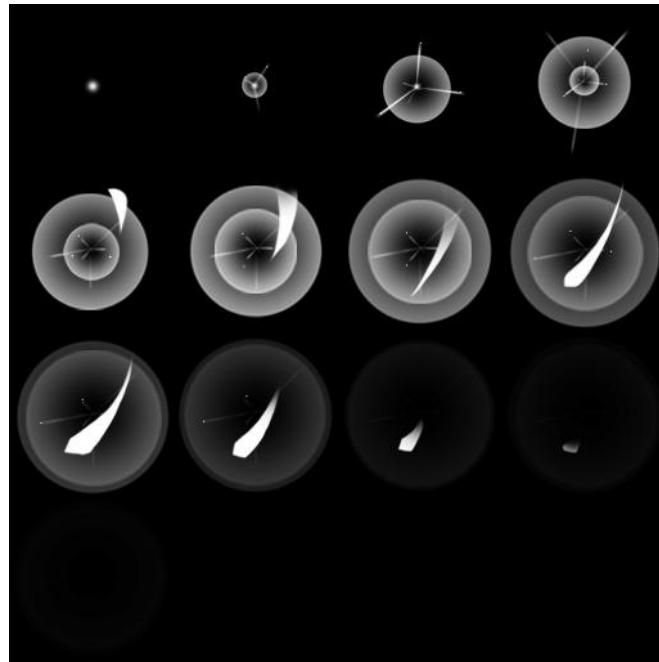


« Figure 95 : traînée de coup d'épée »

Si le coup de l'épée touche une plante, un « **SpriteSheet** » est instancié au niveau du point de collision entre l'épée et la plante pour indiquer au joueur que le coup à réussi. C'est ce que l'on appelle dans le domaine du jeu : la « **sensation de frappe** » [voir figure 96 et 97].



« Figure 96 : sensation de frappe »

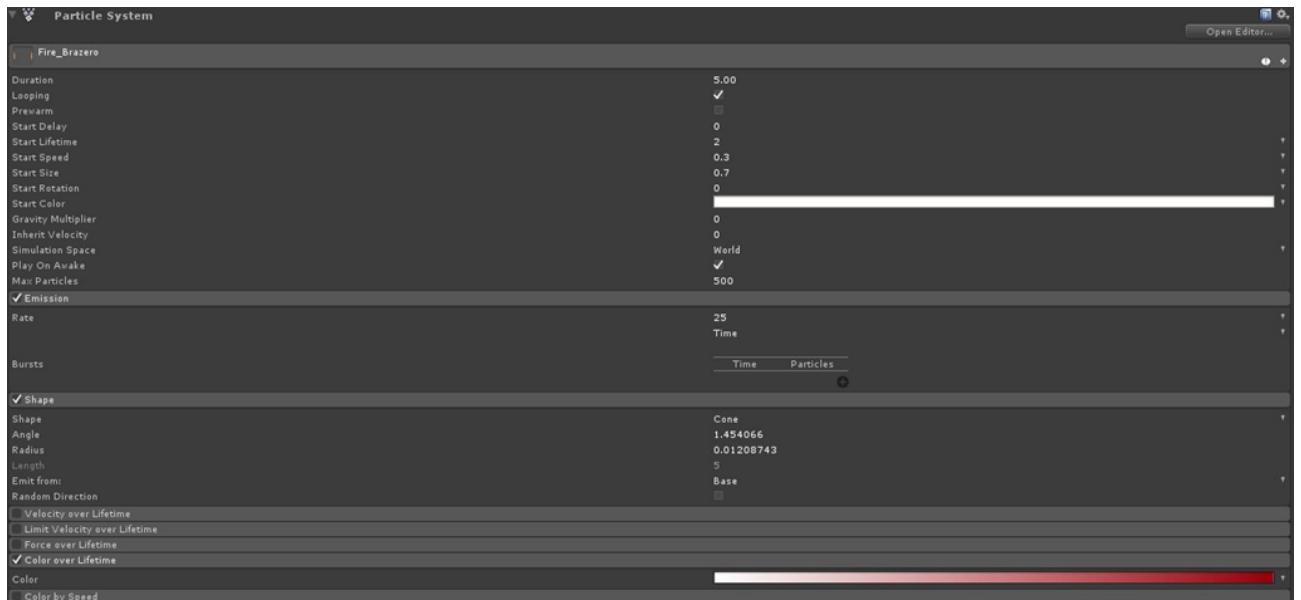


« Figure 97 : exemple de spritesheet »

6.3.3. DustControl

Ce script n'a qu'une utilité : que lorsque les pieds du personnage touchent le sol, un système de particules de poussière apparaît. Il faut savoir que pour les objets instanciés lors des actions comme la poussière ou le « **SpriteSheet** » lors du coup du garçon :

- Ce sont des objets de type « **ParticleSystem** ». On peut donc effectuer les réglages de leurs variables avec l'outil « **Shuriken**⁸² » [voir figure 98].



« Figure 98 : système Shuriken »

- Ils héritent aussi de l'une de nos propres classes telles que « **AutoDestruct** » qui détruit l'objet qui en hérite au bout d'un temps définie dans une variable. Par exemple, une fois la poussière dissipée, l'objet se détruit.
- L'objet hérite aussi évidemment de « **Transform** » pour avoir une position et une rotation sur la scène.
- Pour savoir où l'objet doit être instancié, par exemple la poussière au niveau des pieds ou le « **SpriteSheet** » au niveau de l'épée, on utilise des capteurs de type « **Empty** ». C'est à dire que l'on place au niveau de l'endroit où l'objet doit être instancié, un objet possédant uniquement le « component » « **Transform** ».

6.4. La classe de contrôle du personnage « Umi »

Le personnage « Umi » correspond au prefab « **mage** » et possède les mêmes « components » que celui de « Almas », les deux classes héritent donc des mêmes classes. Cependant, le personnage ne possède pas les scripts de contrôle du garçon mais en possède deux autres similaires : « **FairyControl** » et « **FairyInit** ».

6.4.1. FairyInit

Cette classe est très similaire à « **WarriorInit** » mais agit sur la fille. Elle contrôle donc que seul le joueur qui a instancié le prefab « **mage** » sur la scène de jeu peut le contrôler.

6.4.2. FairyControl

Cette classe fonctionne tout comme « **WarriorControl** » dans le sens où on y retrouve les vérifications de collisions, la fonction « **control()** » et la fonction « **clavier()** ». La fille se déplace dans les aires et donc tous ses déplacements se font en changeant les valeurs des variables héritées de la

⁸² Des couteaux en étoile dont les ninjas se servent pour attaquer à distance.

classe « **Transform** ». La principale différence donc entre « **FairyControl** » et « **WarriorControl** » réside dans les petites fonctions d'actions envoyées en réseau.

- « **Teleportation()** » [voir figure 99] est l'une des plus importantes fonctions d'actions. Elle est déclenchée dans « **clavier()** » lorsque le joueur appuie avec le **bouton droit** de la souris. Elle exécute le pouvoir de téléportation qui correspond à une disparition de la fille et de réapparition instantanée dans un autre endroit. Ce pouvoir est surtout utilisé pour dépasser un mur infranchissable. Pour cela, elle se sert techniquement de la position de la souris à l'écran récupérée grâce à la collision entre un **rayon** lancé depuis la **caméra** vers le **curseur** et un **gameObject** invisible appelé « **viseur** ». La fonction instancie en même temps un système de particules de magie pour indiquer visuellement au joueur que la fille exécute un pouvoir. Ce système de particule, tout comme ceux créés lors des actions du garçon, hérite de « **ParticleSystem** » ainsi que de simples scripts sur le temps de vie de l'objet comme « **AutoDestruct** ».



« Figure 99 : pouvoir – téléportation »

- « **power()** » est la deuxième grande fonction d'action de la fille. Elle est exécutée via « **clavier()** » lorsque la fille appuie sur le **bouton gauche** de la souris. Avec le même procédé que « **Teleportation()** », cette fonction utilise la collision d'un rayon lancé par la caméra pour détecter ce que doit faire la magie. Si le curseur est placé dans le vide, le joueur peut créer un cube, s'il se trouve au dessus d'un cube, le joueur peut alors déplacer le cube et enfin, si le curseur se trouve au dessus d'une plante, la fille lance un projectile magique pour glacer la plante.

6.4. Les pouvoirs de « Umi »

La mise en œuvre des pouvoirs de « Umi » est assez complexe parce qu'ils sont une combinaison de plusieurs objets en même temps :

- Un système de particules pour l'effet visuel.
- Un script de durée de vie maximale pour le projectile de glace.
- Un script de contrôle du pouvoir.
- Un arrêt du contrôle de la fille durant l'utilisation du pouvoir (du à l'orientation RPG).

6.4.1. Tracé du cube

La création d'un cube sur la scène par la fille ne se fait pas avec les fonctions de la classe « **FairyControl** » mais avec une méthode de la classe dont hérite le système de particules de magie instancié dans « **power()** ». En effet, ce système de particule a deux utilité : visuellement, il permet de visualiser ce que le joueur dessine à l'écran, et techniquement, il se sert de la fonction « **creation()** » héritée de notre classe « **MagieControl** ». [voir figure 100]

La fonction « **creation()** » procède de la manière suivante :

- Le tracé d'un cube nécessite quatre sommets et donc quatre points de vérification.
- Toutes les 0.3 secondes, l'ordinateur vérifie la position de la souris. Si la nouvelle position correspond à un tracé de cube on incrémente le nombre de points de vérification. Sinon, si le tracé a été erroné, le nombre de points de vérification revient à zéro.
- Une fois que le bouton gauche de la souris est relâché, si le nombre de points de contrôle est égal à 4, le cube est créé, sinon, rien ne se passe et le système de particules de magie est détruit [voir figure 101].

Le système hérite aussi de « **MagieInit** », qui vérifie que seul le joueur qui contrôle la fille contrôle la magie.



« Figure 100 : dessin de cube »



« Figure 101 : cube créé »

6.4.2. Déplacement du cube

Le cube qui vient d'être instancié est un objet d'une classe qui hérite de « **Rigidbody** », de « **Transform** », de « **MeshRenderer** » et également de deux classes que nous avons créées « **CubeInit** » et « **BoxControl** ».

Les fonctions héritées de la classe « **CubeControl** » en deux principales utilités :

- Déplacer le cube en fonction des mouvements de la souris tant que le bouton droit n'est pas relâché.
- Déplacer « Almas » s'il se trouve au dessus du cube. Pour cela, on vérifie s'il est en collision avec le cube via la fonction « **OnCollisionEnter()** » héritée de « **MonoBehaviour** ». Mais cette vérification est insuffisante, nous vérifions également via la position en « **y** » du component « **Transform** » que le personnage est au dessus du cube.

Lorsqu'on déplace un cube, un système, similaire à la magie utilisée pour tracer le cube, apparaît tout autour [voir figure 102].



« Figure 102 : déplacement du cube »

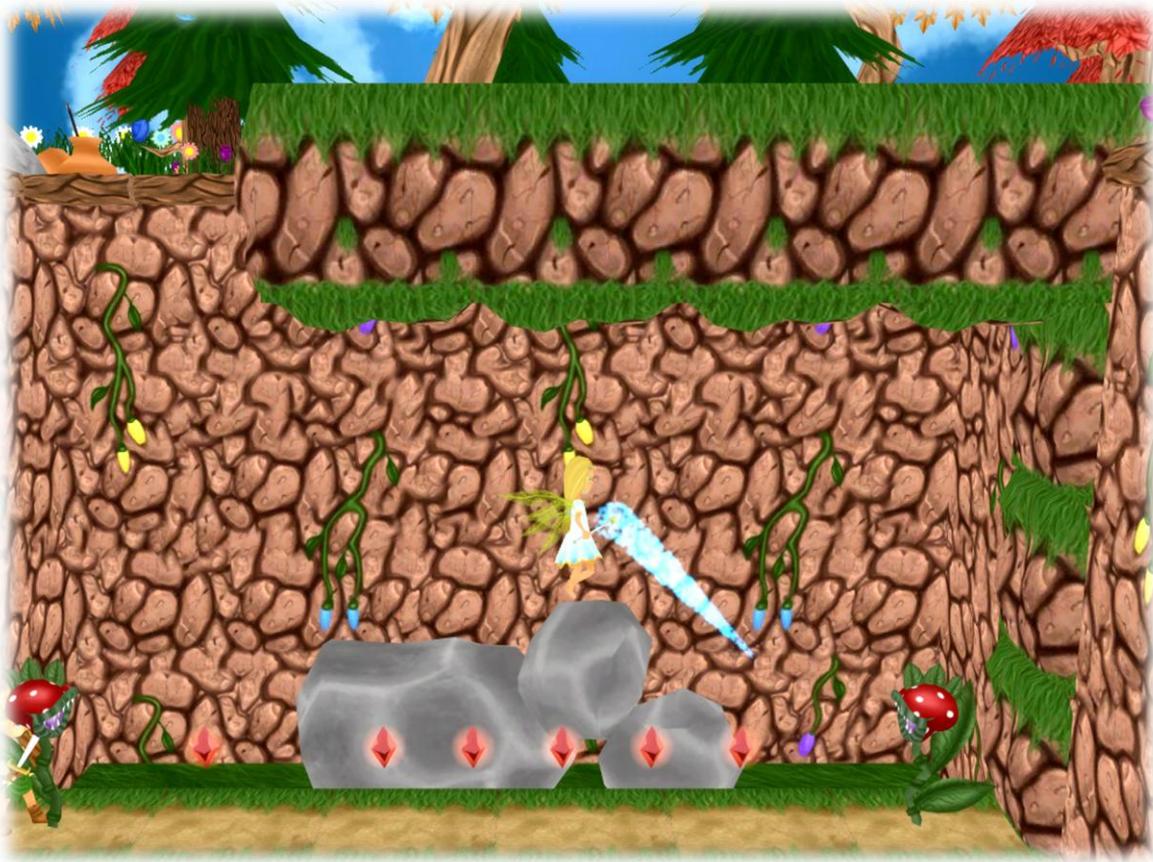
6.4.3. Le pouvoir de glace

Le pouvoir de glace s'active lorsque le curseur se trouve au dessus d'une plante lors du clic de la souris [voir figure 103].

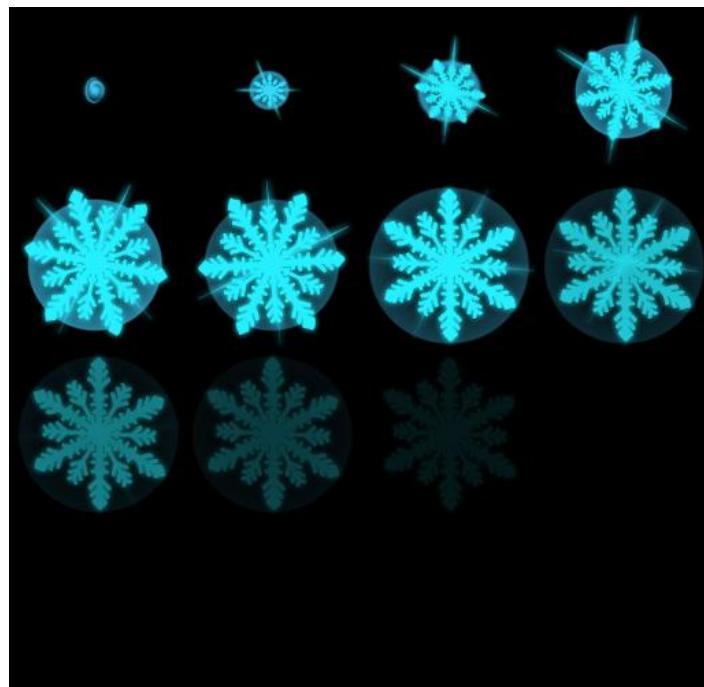
Le « prefab » du pouvoir de glace possède les « **components** » :

- « **BoxCollider** » et « **Rigidbody** » pour détecter la collision avec la plante.
- « **Transform** » ainsi qu'un autre « component » qui change les valeurs des variables du component « **Transform** » pour faire avancer le pouvoir.
- « **ParticleSystem** » pour l'effet visuel d'un projectile de glace. [voir figure 104]

Il faut également savoir que la direction choisie pour que le projectile aille en direction de la plante se fait via la fonction « **LookAt()** » du « component » « **Transform** ». La collision entre le pouvoir et la plante n'est pas vérifiée par un « component » du projectile mais par un de la plante elle-même.



« Figure 103 : projectile de glace »



« Figure 104 : spritesheet effet de glace »

Tout comme pour « Almas », il faut que lorsque la fille lance son pouvoir, le joueur ressent la « **sensation de frappe** ». Ainsi, nous avons ajouté une animation 2D en « **SpriteSheet** » qui se déclenche sur le bout de son sceptre. Le « **SpriteSheet** » est également basé sur le thème de la glace.

7. L'intelligence artificielle

Les plantes volantes possèdent un « component » que nous avons développé et qui correspond à leur intelligence artificielle : la classe « **PlanteMobileControl** ». Cette simulation d'intelligence est utilisée pour que :

- Si la plante est assez proche de la fille, elle l'attaque (change la variable héritée de « Mecanim » pour jouer l'attaque).
- Si elle attaque et qu'elle touche la planche, elle lui enlève de la vie.
- Si elle n'est pas très proche, elle avance vers la fille.
- Si elle trouve un objet sur son chemin, elle l'évite en changeant de direction.

Pour cela, les fonctions de la classe « **PlanteMobileControl** » se basent sur trois grands principes :

- Les variables de valeurs « **aléatoires** » avec la fonction « **Range()** » héritée de la Classe « **Random** ». L'aléatoire est utile pour que plusieurs plantes les unes à côté des autres n'aillent pas à la même vitesse et ne changent pas de directions en même temps.
- La détection de la position de la fille grâce à l'utilisation de « **Raycast** » (rayon capable de détecter des collisions) de la classe « **Physics** » et de la variable « **position** » héritée du component de type « **Transform** ».
- Les durées d'actions permettant à la plante de ne pas être invincible. Cela signifie que la plante ne change de vitesse ou de direction uniquement si le temps est venu de changer de type d'action. S'il n'y avait pas ce principe (hérité du mode de combat RPG), les plantes pourraient éviter toute attaque.

Tout comme les autres personnages, les ennemis sont améliorés visuellement par des effets 2D. Par exemple, les plantes volantes apparaissent dans un nuage de bulle [voir figure 105].



« Figure 105 : apparition en bulles »

7. Un jeu multi-joueurs

La programmation du « **gameplay** » incluant les déplacements des personnages, les classes contrôlant leurs pouvoirs et actions ou encore l'intelligence artificielle d'un ennemi doit être adaptée au fait que deux joueurs jouent sur la même partie. Pour intégrer ce mode multi-joueurs, il fallait tenir compte de quatre grandes contraintes.

7.1. Un serveur et un client

L'un des joueurs doit initialiser un **serveur**, son ordinateur, et l'autre joueur doit connaître son **adresse ip privée** pour s'y connecter en tant que **client**. Lorsque l'on développe cette fonctionnalité de création de serveur, on a le choix de sécuriser la connexion via un mot de passe et l'on peut également choisir un nombre maximal de connexions. Dans notre cas, nous n'avons pas utilisé le mot de passe pour faciliter l'utilisation aux enfants mais nous avons limité la connexion à une seule pour qu'il n'y ait pas plus de deux joueurs sur la scène.

Toutes les fonctionnalités liées à la création ou à la connexion à un serveur sont proposées dans la classe « **Network** » de Unity. Nous avons donc créé un objet sur la scène qui hérite de « **Network** » et d'une classe que nous avons créée : « **Network_ServerConnexion** ». Cette classe se sert des outils hérités par l'objet de type « **Network** » notamment pour créer une partie en initialisant un serveur, mais cette classe ajoute à « **Network** » des fonctionnalités propres à notre projet telles que le fait que seul le serveur choisit le personnage avec lequel il va jouer et que le client obtient donc le personnage disponible.

Voici une liste récapitulative des outils dont nous nous sommes servis :

- « **DontDestroyOnLoad(this);** » exécutée dans la fonction « **Awake()** » donc au démarrage du jeu. Cette fonctionnalité permet au jeu de savoir qu'il ne faut pas détruire l'objet héritant de « **Network_ServerConnexion** » lors du démarrage de la partie. Ainsi, le multi-joueurs sera maintenu.
- « **Network.InitializeServer(1, getListenPort(),this.getUseNate());** », fonction de la classe « **Network** » paramétrée pour une seule connexion et sans utiliser le **NAT** (Network Adresse Translation) car on joue en réseau local.
- « **Network.Connect(getremoteIP(),getListenPort(),getPassword());** » fonction de connexion au serveur paramétrée avec l'adresse IP, le port de lecture et le mot de passe. Dans notre cas, le mot de passe est laissé vide.
- « **Network.Disconnect(200);** », fonction de déconnexion avec un temps d'attente de 200 ms (millisecondes).
- « **OnFailedToConnect(error: NetworkConnectionError)** », fonction permettant de faire une action si la connexion au serveur a échoué. Cela peut arriver si le nombre connexion avec le serveur est déjà atteint ou s'il y a une erreur dans l'adresse IP ou le mot de passe. Cette fois encore, le mot de passe est laissé vide.
- « **OnPlayerDisconnected(player: NetworkPlayer)** », fonction permettant de faire une action si un joueur a quitté le jeu. Généralement, quand un joueur quitte le jeu, on supprime tout simplement son personnage.
- « **OnDisconnectedFromServer(info : NetworkDisconnection)** », fonction permettant d'agir si la connexion avec le serveur a été interrompue. Par exemple, si la connexion est interrompue,

on revient au menu principal.

7.2. « Écouter » et « contrôler »

Dans une partie multi-joueurs, un premier problème se pose. Penons le cas suivant pour mieux expliquer :

- Deux joueurs jouent sur une même partie, mais sur deux ordinateurs différents. Sur la scène se trouve le personnage du premier joueur qui hérite d'une classe « **PlayerControl** » qui permet de déplacer le personnage avec les touches du clavier. Il faut que seul le joueur propriétaire du personnage et donc celui qui l'a instancié sur la scène puisse exécuter les fonctions de cette classe. Dans le cas contraire, les deux joueurs pourraient déplacer le même personnage et en même temps.

Unity permet donc via la classe « **NetworkView** » de savoir qui est le propriétaire de l'objet. En effet, chaque objet héritant de « **NetworkView** » possède un identifiant et l'identifiant de son créateur. Tous nos script de nom « *******Init** » utilisent cette fonctionnalité pour désactiver ou au contraire activer l'héritage à une classe de contrôle de l'objet.

Nous procédons de la manière suivante :

```
if (!networkView.isMine)
{
    this.getComponent(*****Control).enabled = false;
}
```

« Figure 106 : activer / désactiver un héritage »

7.3. Synchroniser les valeurs

Dans une partie multi-joueurs, il faut également que, si un personnage se déplace ou exécute une action visuelle, les changements de valeurs des variables de l'objet soient vus des deux joueurs. Par exemple si un personnage n'a plus de vie , trois changements de variables sont faits : sa vie prend la valeur à zéro, sa position en « z » et sa position en « y » héritées de leur « component » de type « **Transform** » prennent la valeur zéro pour que les personnages reviennent au début du jeu et la variable « **Mecanim** » de passage vers l'animation « **Die** » prend la valeur « 1 ». Il faut que ces trois changements soient envoyés sur le réseau vers le deuxième joueur.

7.3.1. Le problème du « Temps de latence »

Une nouvelle contrainte de programmation vient alors s'ajouter à la simple optimisation du temps de calcul. Le nombre de **FPS** n'est plus le seul problème visuel des joueurs. Il existe à présent un nouveau problème appelé le « **Temps de latence réseau**⁸³ », temps pendant lequel un joueur ne voit pas l'autre personnage en mouvement car les données de synchronisation ne lui sont pas encore parvenues. Ce problème peut être du à deux raisons, soit le nombre d'envoi par seconde des données de synchronisation est insuffisant soit les messages envoyés sont trop lourds et prennent donc beaucoup de temps.

Pour régler ce problème il faut donc :

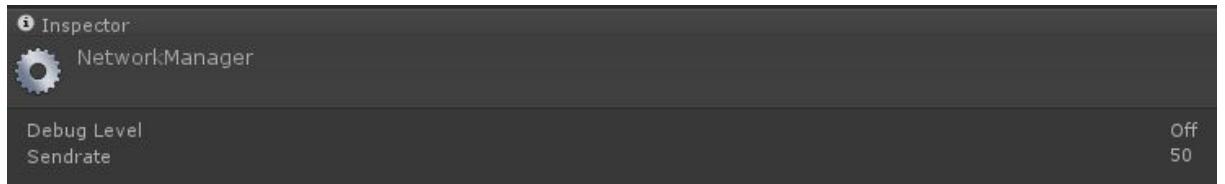
- Ne jamais envoyer des données telles que des « coroutines⁸⁴ » avec « **Yield** », des valeurs importantes comme des sons, objets ou des animations et surtout ne jamais envoyer des

⁸³ Connue aussi sous le nom de « lag ».

⁸⁴ Un traitement de code interrompant la « routine » à laquelle celui-ci est lié tant qu'il n'est pas terminé.

parcours de hiérarchie tels que « `GameObject.Find()` » ou « `this.GetComponent()` ». On n'envoie donc que de simples changements de valeurs qui serviront de déclencheurs. Par exemple, on envoie le nom du `gameObject` dans une variable de type « `String` » plutôt que d'envoyer l'objet lui-même.

- Régler de manière adéquate le « `Send Rate` » dans la fenêtre « `NetworkManager` ».



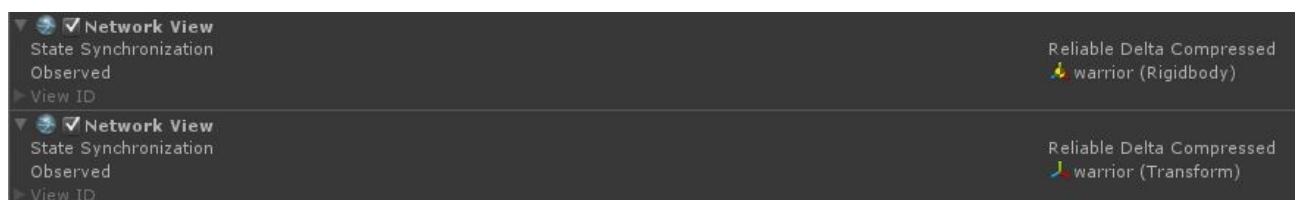
« Figure 107 : sendrate »

- Exécuter le maximum de calcul en local et minimiser le nombre de variables à synchroniser. Par exemple, tout ce qui ne concerne pas les deux personnages mais des objets se déplaçant dans l'environnement doit être calculé chez les deux joueurs de manière indépendante.

7.3.2. Les NetworkView et RPC

Pour envoyer ces variables, et donc les synchroniser chez les deux joueurs, trois outils existent :

- Ajouter à un objet un « component » de type « `NetworkView` » en lui ajoutant le « component ». Après quoi, on choisit quel « component » le « `NetworkView` » doit observer. Par exemple, nos personnages possèdent en « component » plusieurs objets de type « `NetworkView` » pour observer les changements de valeurs de leur « `Transform` » ou de leur « `Rigidbody` ». Cependant, cette méthode a une limite car on ne peut pas observer par exemple les variables d'une classe que l'on a créée soi-même ou encore observer une seule variable d'une classe.



« Figure 108 : NetworkView »

- Ainsi, l'autre méthode est d'ajouter dans nos classes un marqueur « `@RPC` » avant chaque fonction qui doit être envoyée sur le réseau de façon à ce que la fonction soit une « `Remote Procedure Call` ».
- Enfin, il existe des fonctions de la classe « `Network` » telles que « `Network.instantiate()` » ou « `Network.Destroy()` » qui sont nativement envoyées à tous les joueurs.

8. Unity et les bases de données

Enfin, les dernières tâches de ce sprint étaient liées à l'obtention d'un jeu relié à une base de données. Nous souhaitions qu'un joueur démarre le jeu via son pseudo et mot de passe pour qu'il puisse par la suite enregistrer son score via son pseudo.

8.1. Principe général

Le principe général de ce lien est que l'application créée avec Unity s'exécutera comme un « **client lourd** » auprès du serveur web / applicatif qui lui offrira un « **web service** ».

Du côté du client, nous travaillons avec les outils des classes « **WWW** » et « **WWWForm** » qui permettent d'envoyer vers une « **URL** » des variables en « **\$_POST** ». Il faut que l'URL corresponde à un script **PHP** dans lequel le traitement de vérification de l'existence du couple login / password dans la base de données aboutit sur un « **echo** » qui sera reçu du côté client.

8.2. La base de données SQL

Notre base de données est constituée de trois tables : une pour les scores, une pour les parties jouées et une pour les joueurs inscrits [voir figure 109].

```
drop database if exists abdevfr_theTwins;

create database abdevfr_theTwins;
use abdevfr_theTwins;

CREATE TABLE IF NOT EXISTS `userTwins` (
    `id` int(10) UNSIGNED NOT NULL AUTO_INCREMENT,
    `pseudo` varchar(30),
    `password` varchar(30) NOT NULL,
    CONSTRAINT PKuser PRIMARY KEY (id)
);

CREATE TABLE IF NOT EXISTS `partieTwins` (
    `idPartie` int(10) UNSIGNED NOT NULL AUTO_INCREMENT,
    CONSTRAINT PKpartie PRIMARY KEY (idPartie)
);

CREATE TABLE IF NOT EXISTS `scoreTwins` (
    `idScore` int(10) UNSIGNED NOT NULL AUTO_INCREMENT,
    `idPartie` int(10) NOT NULL,
    `idUser` varchar(30) NOT NULL,
    `score` int(15) NOT NULL DEFAULT 0,
    CONSTRAINT PKscore PRIMARY KEY (idScore),
    CONSTRAINT FKscoreUser FOREIGN KEY (idUser) REFERENCES userTwins (id),
    CONSTRAINT FKscorePartie FOREIGN KEY (idPartie) REFERENCES partieTwins (idPartie)
);
```

« Figure 109 : requêtes SQL - création d'une base de données »

8.3. Le script exécuté côté client

Du côté client, la classe « **GetUser** » de la couche « **Accès aux données** » possède les fonctions interagissant avec la base de données.

Dans ce script, on crée un objet de type formulaire avec la classe « **WWWForm** » dans lequel on ajoute les valeurs des pseudos et mots de passe avec la fonction « **AddField()** » que l'on envoie en

ligne et on récupère la réponse à l'aide de la fonction « **text()** » de la classe « **WWW** ». « **text()** » récupère en fait le message écrit dans le script PHP avec la fonction « **echo()** ».

Si le message reçu a la valeur « **true** » en chaîne de caractères, alors la connexion est validée et le joueur peut démarrer le jeu.

8.4. Les scripts exécutés côté serveur et sécurité de la base de données

8.4.1. Connexion.php

Le premier script PHP n'est qu'une simple liste des « **DEFINE** » utilisés pour la connexion entre le PHP et la base de données. On y trouve donc les valeurs des variables du nom de l'utilisateur, du nom de la base de données, du mot de passe et la base de données.

8.4.2. Protection.php

Ce script inclut les classes PHP correspondant aux tables de la base (les utilisateurs, les parties et les scores) ainsi que le script « **Connexion.php** » à l'aide de la fonction PHP « **require_once()** ».

- Ensuite, dans ce script on trouve une fonction exécutée sur les variables reçue en « **POST** » pour contrer les tentatives d'injection de code :

```
public static function protect($arg)
{
    $conn = mysqli_connect(SERVER, USER, PASSWORD, DB_NAME);
    $return=mysqli_real_escape_string($conn, strip_tags($arg));
    mysqli_close($conn);
    return $return;
}
```

« Figure 110 : script contre les tentatives d'injection de code »

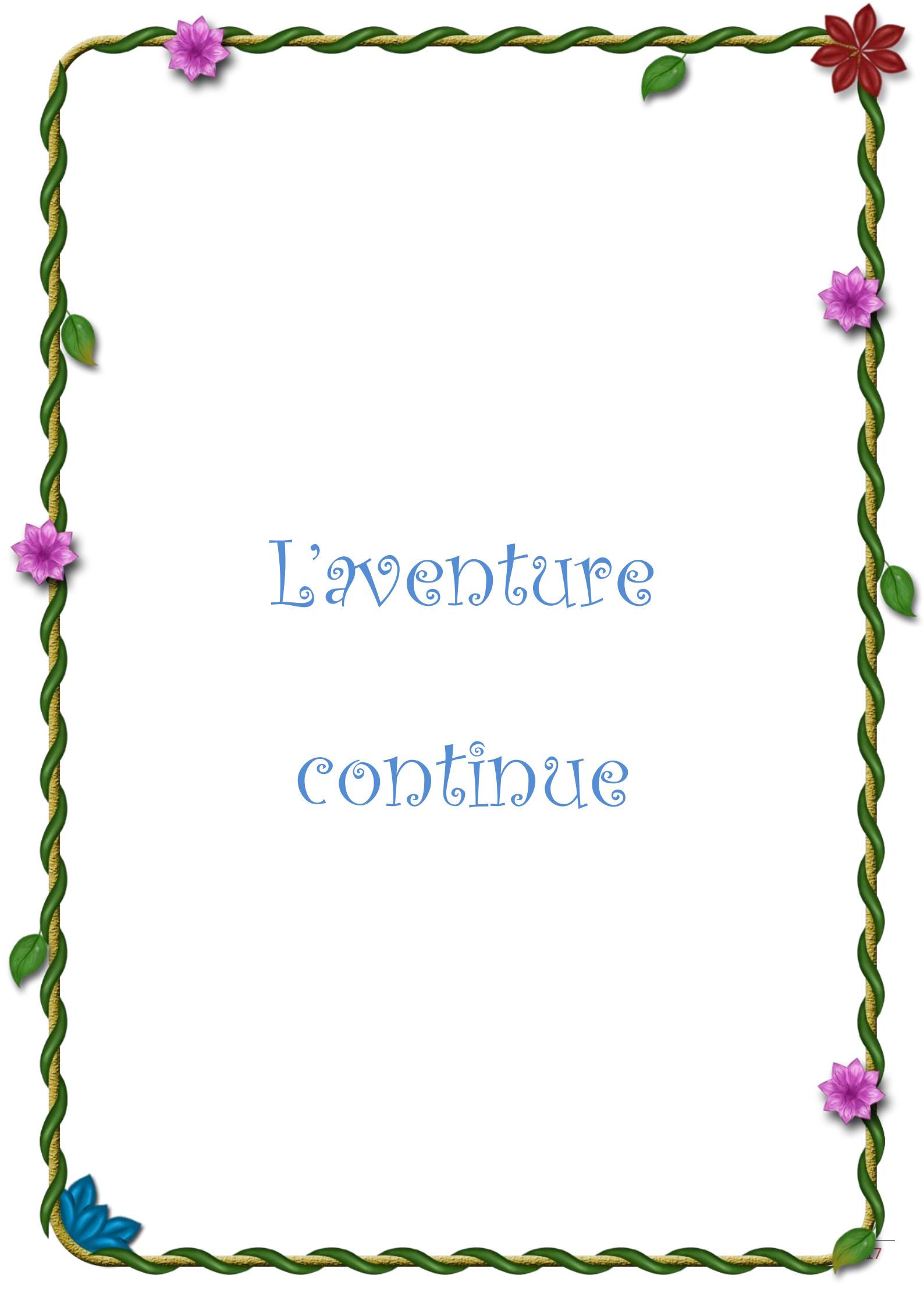
- Enfin, pour une meilleure sécurité de la base de données, on trouve dans ce script une fonction qui **chiffre** les mots de passe. Ainsi, quand on se connecte, on chiffre la variable reçue en **POST** et on vérifie que le résultat chiffré correspond au mot de passe enregistré dans la base de données. On chiffrera donc également le mot de passe lors de l'inscription. Pour cela, nous nous sommes inspirés de l'algorithme de « **Jules Cesar** » avec une clé de chiffrement de 4. Pour cela, notre fonction « **crypte()** » utilise la fonction PHP « **strtr()** »

8.4.3. GetUser.php

Ce dernier script est celui qui reçoit les variables **POST** envoyées par le **client lourd** et renvoie un **echo** en réponse. Il inclut « **Protection.php** » et se sert des fonctions de sécurité de celui-ci.

Le mécanisme de vérification utilise les fonctions « **mysqli** » telles que **mysqli_connect()** ou **mysqli_query()**, plus récentes que « **mysql** » pour éviter les problèmes d'incompatibilité.

Enfin, ce script apporte une vérification de sécurité supplémentaire en plus de celles incluses via « **Protect.php** ». Le script côté client envoie en plus des variables de connexion, une autre variable appelée « **twins** ». Ainsi, si le script PHP ne reçoit pas cette variable, alors la tentative de connexion est faite à partir d'un autre script que celui que nous avons développé. Il ne faut donc rien exécuter !



L'aventure
continue

II. Sprint 2 :

Création de l'environnement et des énigmes



1. Tableau prévisionnel du sprint

- *Date de début du sprint* : Lundi 3 mars 2014
 - *Date de fin de sprint* : Lundi 31 mars 2014
 - *Temps estimé en heures* : 260 heures
 - *Échelle de mesure* :
 - Une journée égale 8 heures de travail (de 9h->13h puis de 14h->18h)
 - On compte dans le tableau, les heures restantes en fin de journée (donc après 8 heures de travail).
 - *Objectifs du sprint* : Création de l'environnement (map) du jeu qui répondra à un scénario et des énigmes que les deux joueurs devront résoudre pour finir ce scénario.

1.1. Première partie du mois

« Tableau 4: première partie du mois - sprint 2 »

1.2. Deuxième partie du mois

Nom :	Priorité	Temps	19	20	21	22	23	24	25	26	27	28	29	30	31
Texture et UV des objets dénigmes	46	8h	0	0	0	0	0	0	0	0	0	0	0	0	0
Level Design et composition finale de la scène	47	24h	24	16	8	0	0	0	0	0	0	0	0	0	0
Scripts de déroulement du temps et du score	48	8h	0	0	0	0	0	0	0	0	0	0	0	0	0
Délimitation de l'espace de jeu	49	8h	8	8	8	8	0	0	0	0	0	0	0	0	0
Scripts de récupération des diamants	50	4h	4	4	4	4	4	0	0	0	0	0	0	0	0
Scripts de collision avec les eaux	51	4h	4	4	4	4	4	0	0	0	0	0	0	0	0
Scripts de lénigme des ponts	52	4h	4	4	4	4	4	4	0	0	0	0	0	0	0
Scripts de lénigme de la balançoire	53	4h	4	4	4	4	4	4	0	0	0	0	0	0	0
Scripts des plateformes mobiles	54	4h	4	4	4	4	4	4	4	0	0	0	0	0	0
Scripts des flèches	55	4h	4	4	4	4	4	4	4	0	0	0	0	0	0
Scripts de la zone d'éboulement du rocher	56	4h	4	4	4	4	4	4	4	4	0	0	0	0	0
Scripts de la zone des blocs de pierres balançoires	57	4h	4	4	4	4	4	4	4	4	0	0	0	0	0
Scripts de la zone de fin de niveau	58	4h	4	4	4	4	4	4	4	4	4	4	0	0	0
Ajout des effets de particules (eau, poison, feu...)	59	8h	8	8	8	8	8	8	8	8	8	8	4	0	0

« Tableau 5 : deuxième partie du mois - sprint 2 »

2. Conception et création graphique

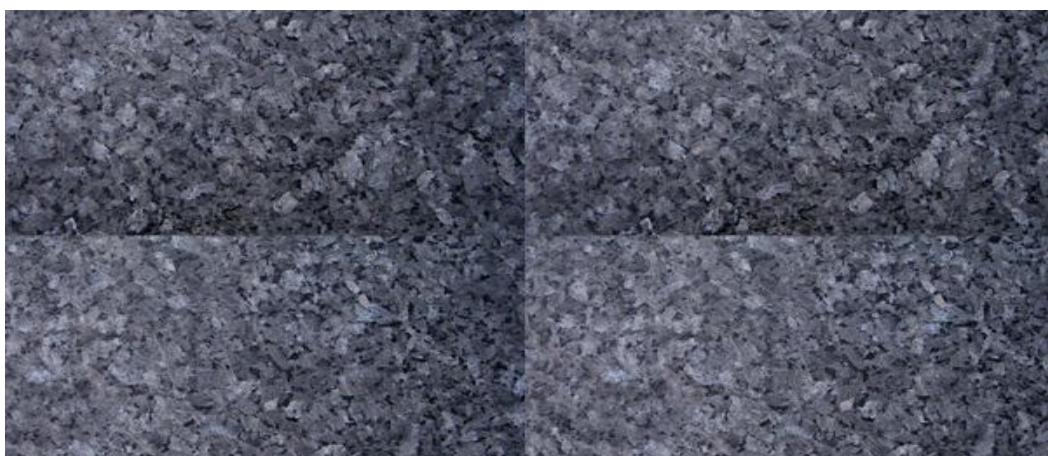
Les étapes de la création graphique des objets de l'environnement sont les mêmes que celles utilisées pour les personnages : **conception 2D, modélisation** puis « **texturing** ».

Cependant, bien que beaucoup de contraintes propres à la création des personnages ne soient pas importantes dans la création de l'environnement, d'autres contraintes doivent être observées :

- Les objets de l'environnement sont très nombreux sur la scène, il faut limiter le nombre de polygones bien plus que les personnages. De plus, ils sont moins importants et parfois très petits. On essaie donc d'optimiser le calcul des polygones de l'environnement.
- Pour cette même raison d'optimisation, on texture sur un même fichier plusieurs objets 3D. On gagne en ressources car pour par exemple cinq objets 3D différents, la carte graphique ne tient compte que d'une seule texture utilisée sur les cinq objets.
- Un même objet sera utilisé plusieurs fois et dans divers endroits, il faut donc qu'il puisse s'adapter à l'endroit dans lequel il va être placé. Par exemple l'objet 3D « pont » peut être utilisé pour que les personnages puissent traverser un petit lac puis un grand lac. Il faut donc qu'il soit modélisé en « **pièces détachées** » qui ne seront assemblées que dans la phase de « **level design** ». Ce principe de la réutilisabilité est le « **modular** ». De plus, modéliser en « **pièces détachées** » permet de minimiser l'impression de répétition des objets par le joueur.

La meilleure illustration pour expliquer à quoi correspond le « **modular level design** », est le jeu pour enfant : « **Lego** ». Dans ce jeu, l'enfant possède une collection de pièces de formes simples qui peuvent s'emboîter les unes dans les autres pour construire un objet plus complexe.

La « coupure » est la visible séparation entre deux objets 3D. Elle peut être due aux formes des objets qui ne forment pas une suite ou à leurs textures qui n'en forment pas une seule lorsqu'elles sont l'unes à côté de l'autre. Pour que la « coupure » entre deux objets emboîtés ne se voie pas, il faut que leurs textures soient continues d'un côté et de l'autre. Ce type de texture capable de se répéter sans créer de coupure est appelé « **Tiled textures** ». Voici une illustration [voir figure 111] du problème qui arrive lorsque l'on duplique un objet dont la texture n'est pas créée de manière à ce qu'elle se répète sans coupures.



« Figure 111 : texture non tiled »

2.1. Conception des objets du décor

Nous avons choisi de concevoir les objets du décor sur un même fichier de manière à créer des formes homogènes les unes avec les autres.

Nous avons commencé par lister les types d'objets potentiellement utilisables pour créer un univers forestier : des arbres, des fleurs, des champignons, des bûches ou encore des rochers. Puis, nous avons imaginé qu'il faudrait des animaux pour créer un peu de mouvements et avons alors pensé aux papillons. Enfin, pour retranscrire le scénario dans le jeu, il nous fallait deux villages. Ainsi, nous avons pensé à créer des objets manufacturés par des hommes tels que des tentes, des braseros, des ponts, des planches ou encore des petits éléments comme des flèches ou des jarres...

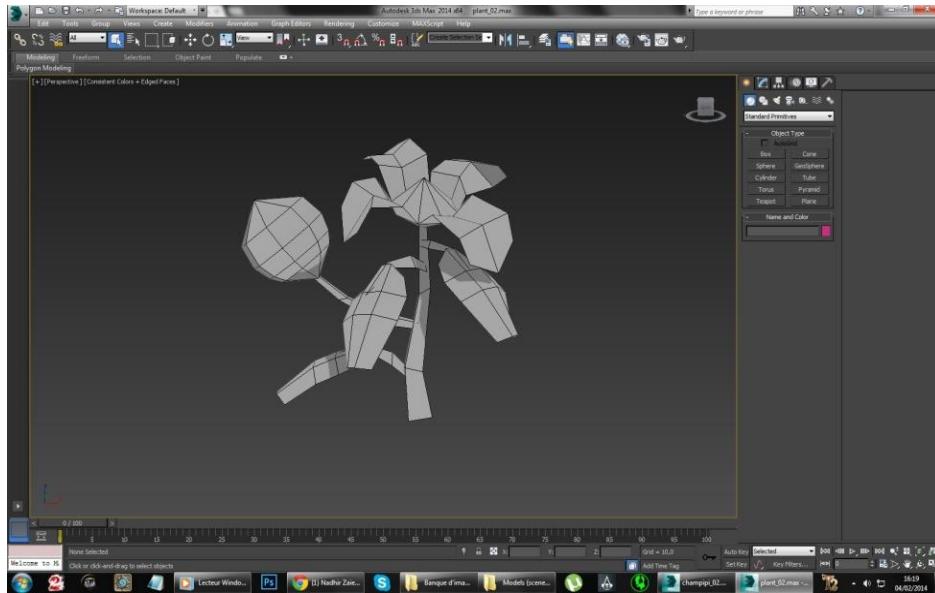
À part les objets du décor, il faut aussi créer le sol sur lequel tout cela va reposer. Ce sont les composants du sol que nous avons choisis de concevoir, capables de s'emboîter tels des « **Lego** ».



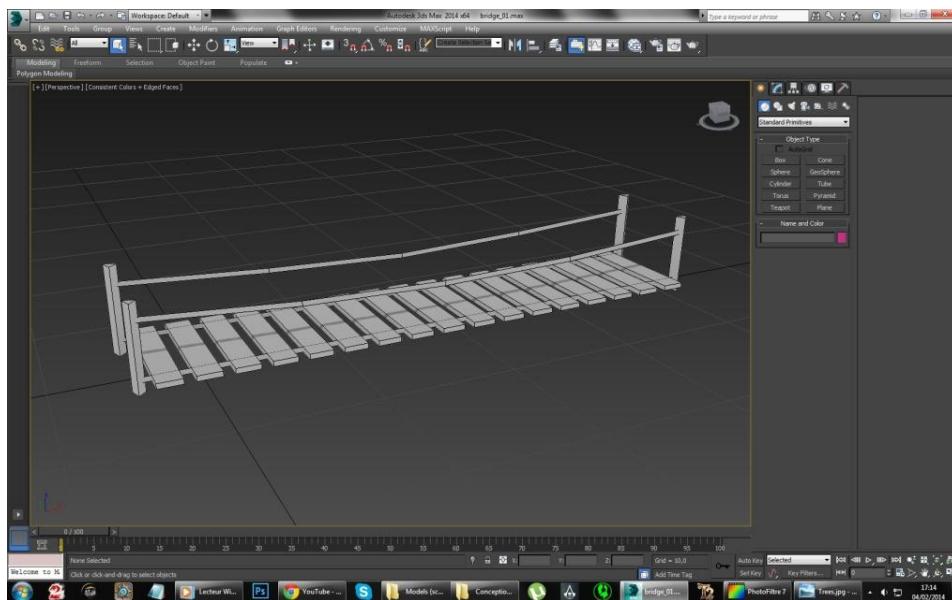
« Figure 112 : conception des éléments du décor »

2.1 Modélisation des objets du décor

Pour les objets du décor nous n'avons pas utilisé les « **blueprints** ». Le résultat est donc moins fidèle aux conceptions que les personnages mais cela permet en contre partie de modéliser beaucoup plus vite [voir figure 113 et 114].



« Figure 113 : modélisation sans blueprint 1 »



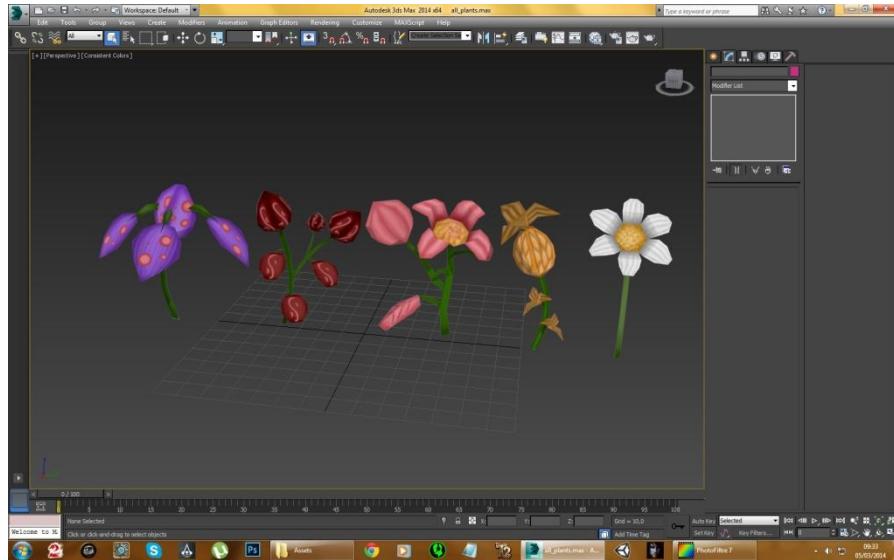
« Figure 114 : modélisation sans blueprint 2 »

2.2. Textures des objets du décor

- Voici un exemple de texture regroupant plusieurs objets 3D pour l'optimisation de la mémoire [voir figure 115] et le résultat final une fois intégrée [voir figure 116].



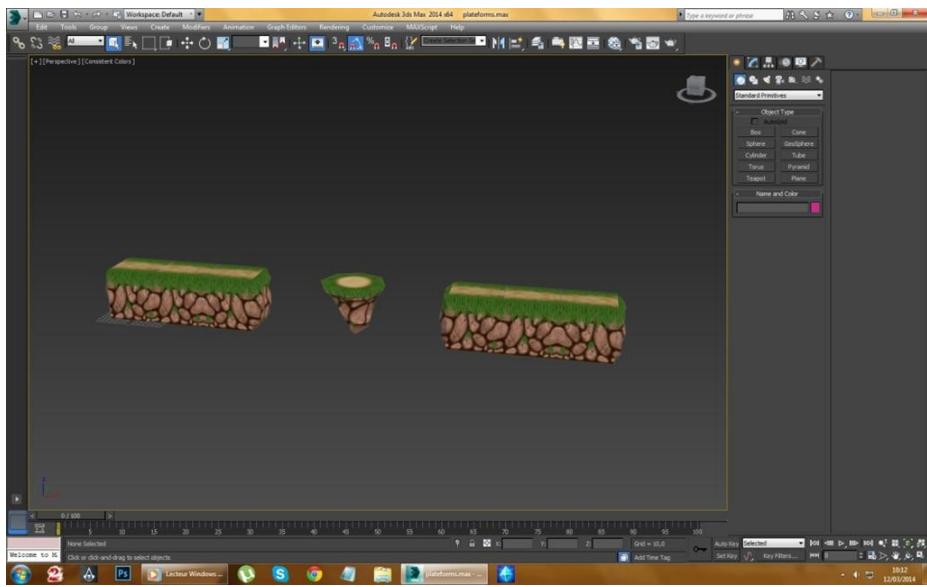
« Figure 115 : texture regroupant plusieurs objets 3D »



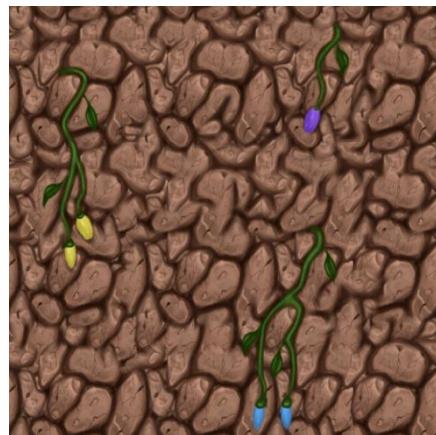
« Figure 116 : résultat final après intégration de textures »

- Les objets « **modulables** » du sol sont au nombre de six:
 - Le chemin de sable sur lequel les personnages se déplacent,
 - Les deux extrémités du chemin qui apparaissent par exemple avant un pont.
 - Des blocs de pierres flottants dans les airs,
 - le sol de recouvert de verdure autour du chemin principal. C'est sur ce sol que se trouveront les arbres et autres plantes.
 - Enfin, des façades de pierres.

Ils possèdent tous des textures « **tiled** » [voir figure 117 et figure 118].



« Figure 117 : objets modulables du sol »



« Figure 118 : texture tiled »

2.3. Ajout du ciel

Choisir un ciel à la fois comique et magique n'a pas été facile [voir figure 119], nous avons hésité à créer quelque chose de plus classique avant d'avoir l'idée finale d'un ciel en « **volute** » [voir figure 120].



« Figure 119 : conception de ciel »

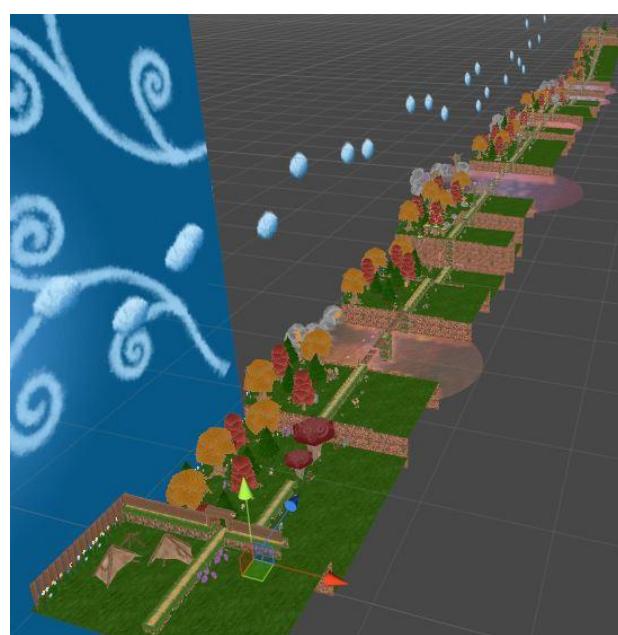


« Figure 120 : ciel en volute »

3. Composition de l'environnement

Une fois toutes les modélisations prêtes, il faut les importer dans Unity comme nous l'avons fait pour les personnages. Les modélisations du décor ne possédant pas d'animations, nous supprimons les « components » d'animation lors de l'importation pour éviter la perte inutile de mémoire.

Une fois les modèles importés, nous pouvons composer. Le principe de l'univers d'un jeu 2.5D est qu'il s'étale dans la longueur et non dans la largeur comme un parchemin. La partie du monde se trouvant derrière la caméra ou celle en dessous du sol sont vides car elles sont invisibles dans le jeu. Cet aspect d'un monde en parchemin n'est pas ressenti par le joueur.



« Figure 121 : composition de l'environnement »

3.1. Amélioration visuelle

Nous nous sommes ensuite servis des fonctionnalités offertes par **Unity** pour embellir le monde que nous avions créé :

- Le « **component** » « **Halo** » permet de créer une aura de lumière autour d'un objet. Avec cet outil, nous avons amélioré les papillons, les diamants et certains systèmes de particules [voir figure 122].



« Figure 122 : halos »

- L'outil « **Shuriken** » avec lequel nous avons ajouté des effets de particules tels que les bulles au dessus des eaux, les feux éclairant les portes des villages ou encore du poison qui sort des vignes.
- Les « **components** » de catégorie « **Image Effect** » qui permettent d'ajouter des effets ou des filtres à la caméra. Ainsi, pour améliorer la qualité de l'image, nous nous sommes servis de ces filtres pour ajouter du contraste (**ContrastStretchEffect**) et un effet de surbrillance (**Bloom**).
- Nous avons aussi ajouté une animation sur certaines modélisations 3D telles que deux « Grands Champignons » que l'on rencontre au début de la partie. Cette animation correspond à un effet de rebondissement déclenché lorsque « Almas » saute sur le champignon. Ainsi le personnage ressent réellement l'impression que l'objet 3D possède les caractéristiques physiques d'un champignon.

3.2. Délimitation de l'espace de jeu et optimisation des colliders

Pour que l'environnement puisse réellement être utilisé, il faut délimiter sa taille en ajoutant des barrières infranchissables. Sans cette précaution, le personnage de « Umi », qui vole et se téléporte pourrait par exemple monter sans limite dans le ciel ou descendre en dessous du sol en se téléportant. Le joueur sortirait ainsi du cadre du jeu et pourrait alors voir à l'écran certaines zones vides ou non travaillées. Nous ne souhaitons pas cela et avons donc créé ces limites en plaçant tout autour du monde jouable des objets invisibles mais possédant le « **component** » « **BoxCollider** ». Au final, les objets possédant un « **component** » de collision sont :

- Les diamants que l'on récupère possèdent un « **BoxCollider** » de type « **Trigger** » pour détecter la collision avec « Almas »,
- Tous les sols et façades possèdent des « **BoxCollider** » de type « **Non Trigger** » pour que les personnages et les ennemis ne puissent passer à travers,
- Les personnages et les ennemis possèdent des « **BoxCollider Non Trigger** ».

On remarque qu'il y a beaucoup de « Colliders » sur la scène. Hors, ce « **componoent** » faisant appel à la physique, il est très lourd en ressources. Il convient donc d'en minimiser le nombre présent sur la scène. Ainsi, nous avons par exemple choisi que sur toute une rangée droite d'objets formant le sol, un seul de ces objets posséderait le « **component** ». Ainsi on agrandit la taille du « **BoxCollider** » par rapport à la taille de l'objet de manière à ce que le « **BoxCollider** » couvre toute la rangée droite de sol. Ainsi, on peut diviser jusqu'à dix fois le nombre original d'objets de collision [voir figure 123].



« Figure 123 : optimisation au niveau des colliders »

Nous avons également établi quelques règles supplémentaires quant aux « **colliders** » pour améliorer le réalisme des actions dans le jeu :

- Nous avons crée une liste de « **Tags** » dans le « **Tag Manager** ». Ainsi, si nos personnages entrent en collision avec un « **gameObject tagé** » en tant que « **water** », ils perdent toute leur santé. Le personnage « Almas » ne peut sauter que s'il est en collision avec un objet tagé « **sol** ». Lorsque le personnage « Almas » entre en collision avec un objet tagé « **facade** » ou « **facadeSol** » et qu'il n'est pas en collision avec le sol, il ne peut sauter et une force est appliquée pour qu'il glisse vers le bas.
- Nous avons ensuite créé une liste de « **Layers** » (variable également liée au « **component** » « **Transform** »). Organiser nos objets dans différents « **Layers** » est très utile pour définir par la suite si un objet d'un layer A peut créer des collisions avec un objet d'un layer B. Par exemple, nous ne souhaitons pas, pour ne pas gêner les joueurs lors de la partie, que « Almas » puisse entrer en collision avec « Umi » [voir figure 124].

	Default	TransparentFX	Ignore Raycast	Water	cube	fairy	pont
Default	✓	✓	✓	✓	✓	✓	✓
TransparentFX	✓	✓	✓	✓	✓	✓	✓
Ignore Raycast	✓	✓	✓	✓	✓	✓	✓
Water	✓	✓	✓	✓			
cube					✓		
fairy	✓	✓					
pont	✓						

« Figure 124 : layers de collisions »

4. Développement des énigmes et du temps

L'expérience que nous voulions que le joueur acquière ne se limite pas à courir dans la forêt et se battre contre les plantes qu'ils y trouveront. Nous souhaitons également que le saut du garçon et les pouvoirs de la fille soient utilisés pour franchir des énigmes. Les énigmes seront assez simples car il ne faut pas oublier l'âge des joueurs qui devront les résoudre. Elles seront également en accord avec le monde forestier.

4.1. Des ponts pas très solides

La première énigme, construite avec le principe de la fonction « **AddForce()** » de la classe « **Rigidbody** » est sans doute la plus facile à résoudre. Nous avons placé un objet de collision invisible et « **Trigger** » devant les ponts de bois. Quand l'objet entre en collision avec le personnage « **Almas** », tous les objets possédant le « **component** » « **ChuteElementControl** », que nous avons développés, tombent dans le vide [voir figure 125]. Ainsi le pont s'effondre juste avant l'arrivée de « **Almas** ». La fonction de chute de « **ChuteElementControl** » agit sur les valeurs des variables des instances de « **Rigidbody** » pour faire tomber les éléments du pont. Nous avons ensuite réglé manuellement les valeurs des forces appliquées sur chacun des éléments du pont. Ainsi, pour obtenir le résultat le plus réaliste, la première planche tombe avec la force la plus forte et la dernière avec la force la plus faible. Pour passer le vide qui existera après la chute du pont, « **Umi** » doit transporter « **Almas** » sur un cube jusqu'à l'autre rive du lac.



« Figure 125 : pont qui s'effondre »

4.2. La balançoire, un interrupteur

Sur le chemin, les deux personnages se trouveront bloqués par deux grandes vignes. Ils ne peuvent sauter au dessus car leurs fleurs dégagent une vapeur violette empoisonnée. « Umi » doit se téléporter de l'autre côté et faire descendre les vignes dans le sol à l'aide d'un interrupteur. L'interrupteur est en fait une simple balançoire en bois et le joueur doit comprendre qu'il doit basculer cette balançoire en posant un lourd cube au dessus.

Cette fois encore nous avons travaillé avec la classe « **Rigidbody** ». Sans aucune programmation, mais en réglant la « **masse** » du cube plus forte que celle de la balançoire, nous obtenons le résultat souhaité. Quand le cube est posé sur la balançoire, elle se penche sous le poids du cube. Par la suite, il faut bloquer les autres mouvements de la balançoire tels que des rotations non souhaitées.

Enfin, nous avons créé un objet de collision de type « **Trigger** » en dessus de la balançoire. Si elle penche avec le poids du cube, la collision sera faite. Ainsi cet objet de collision possède le script qui ouvre les vignes s'il entre en collision avec la balançoire. L'ouverture se fait de la manière suivante :

- La vigne possède un « **component** » « **Transform** ». On applique une descente en changeant les valeurs des variables du « **Transform** » telles que la « **position y** » et on ajoute à cela une rotation autour de l'axe « **y** ». Pour cela on utilise les fonctions « **Translate()** » et « **Rotate()** » de la classe « **Transform** ».
- On ajoute un système de particules de poussière autour du bas de la vigne. Cela sert à cacher la coupure produite lorsque la vigne rentre dans le sol. [voir figure 126]



« Figure 126 : balançoire »

4.3. Attention aux pierres

Pour éviter le sentiment que le monde n'est qu'une simple ligne droite, nous avons ajouté une montée et une descente. Nous avons ensuite ajouté un dangereux piège dans cette descente pour que le joueur remarque le changement et qu'il s'en souvient. Pour créer un lien visuel entre toutes les énigmes, le piège est encore un jeu de physique créé avec à l'aide de la classe « **Rigidbody** ». Quand le garçon entre en collision avec le sol de la descente, un grand rocher de pierre commence à tomber sur lui. Si « Almas » entre en collision avec le rocher alors qu'il roule il se fait écraser. S'il le touche d'en haut, de derrière ou si le rocher n'est pas en mouvement, « Almas » ne sera écrasé.

Pour cela, la force propulsant le rocher dans la descente est créée avec la fonction « **Addforce()** » de la classe « **Rigidbody** ». La vérification sur la vitesse du rocher pour savoir s'il peut, ou non, écraser « Almas » se fait avec la valeur de la variable « **velocity** » du « component » « **Rigidbody** ». Toutes ces vérifications se font dans la fonction « **OnCollisionEnter()** » du **component** « **WarriorControl** » [voir figure 127].



« Figure 127 : éboulement de rocher »

4.4. Un lac piégé

L'un des grands classiques des jeux de plateformes est de se trouver dans une zone où un objet dangereux apparaît à un intervalle régulier de temps. Le joueur doit alors réguler sa vitesse en prévoyant à quel moment l'objet va apparaître pour ne pas le toucher. Dans notre cas, un mécanisme de lancer de flèches est placé au fond d'un lac. Les flèches apparaissent via la fonction « **Instantiate()** » de la classe « **Network** » et sont déplacées vers le haut avec la fonction « **Translate()** » de la classe « **Transform** ». Pour ajouter un effet visuel de vitesse et de dynamisme, les flèches sont suivies d'une traînée lumineuse.

Chaque flèche créée utilise de la mémoire, il convient donc de les détruire une fois qu'elles ne sont plus visibles par les joueurs. Cela arrive lorsqu'elles ont atteint la limite supérieure de l'espace jouable de la scène [voir figure 128].



« Figure 128 : lac piégé »

4.5. Aucun moyen de passer

Le niveau de difficulté des énigmes étant croissant au fil du jeu, les joueurs arrivent dans une zone où le moyen de passer n'est pas visible. Cette zone est un lac sans aucun support pour que « Almas » puisse marcher et trop long pour pouvoir le faire passer à l'aide d'un cube. « Umi » doit en fait utiliser son pouvoir de glace pour geler l'eau par endroit. Une fois gelée, « Almas » peut marcher dessus. Cependant, cela ne dure pas. Un bloc ne reste glacé que quelques secondes avant de fondre dans l'eau.

Techniquement, quatre objets de collision invisibles et de type « **Trigger** » sont placés dans le lac. Ils sont disposés de manière à ce que quelque soit l'endroit où la fille lance son pouvoir de glace, le pouvoir touchera l'un des quatre objets de collision. S'il y a collision, un « **GameObject** » en forme de bloc de glace est instancié sur la scène là où il y a eu collision.

Ce bloc de glace possède lui aussi un « **component** » que nous avons développé : « **blockGlacControl** ». Un objet de type « **blockGlacControl** » possède trois variables :

- **timeToLive** possède la valeur zéro lors de l'apparition et est incrémenté avec le temps dans la fonction « **Update()** » héritée de « **MonoBehaviour** », quand **timeToLive** atteint la valeur « quatre secondes », la variable booléenne « **OnLive** » devient « **false** ».
- **TimeToDie** correspond au temps durant lequel le bloc descend dans l'eau avant de disparaître totalement. Cette variable commence à s'incrémenter lorsque « **OnLive** » est devenu « **false** ». Si **TimeToDie** n'a pas encore atteint la valeur « deux », le bloc descend. Après avoir atteint cette valeur, le bloc disparaît avec la fonction « **Destroy** » envoyée en RPC [voir figure 129].



« Figure 129 : lac gelé »

4.6. Des joyaux presque inaccessibles

Une autre vigne bloque le passage. Celle-ci ne s'ouvre qu'après avoir récupéré tous les joyaux qui se trouvent avant elle. On ne teste donc pas la réflexion mais les réflexes du joueur qui contrôle le personnage « Almas ».

Techniquement, chaque fois que « Almas » récupère un joyau, la collision provoque une incrémentation du nombre de joyaux récupérés. Ce nombre correspond à la valeur d'une variable dans un « component » de la vigne.

4.7. Le temps, un système de score.

Le réel challenge pour les joueurs n'est pas tant de trouver la solution aux énigmes, c'est surtout de la trouver le plus vite possible.

4.7.1. Calcul du score :

Le personnage appartenant au joueur **client** est le seul à posséder le « component » « **CalculTime** » que nous avons créé dans la couche métier. Cette classe contient une fonction d'incrémentation du score en fonction du temps ;

Ainsi le score commence à s'incrémenter qu'une fois que les deux joueurs sont sur la scène. On vérifie le joueur avec les valeurs des variables « **isServer** » et « **isClient** » de la classe « **Network** ».

Le score s'incrémentera dès l'apparition du personnage avec « **DeltaTime** », une variable de la classe « **Time** ». Après quoi le score est envoyé en réseau au joueur « **serveur** » par « **RPC** ». Le score est calculé en secondes mais est affiché sous sa forme « minutes : secondes ». On utilise pour ça une fonction que nous avons créée et qui divise le score par 60 et ajoute le nombre de secondes qui reste.

4.7.2. Enregistrement du score :

Deux types d'enregistrements sont effectués par rapport au score :

- Un enregistrement local du score sans tenir compte du pseudo. Cet enregistrement est automatique en fin de partie et ne demande pas de confirmation auprès du joueur. Pour cela on utilise les fonctions statiques de la classe « **PlayerPrefs** ».
- Un enregistrement en ligne avec demande de confirmation du joueur. Pour cela on utilise le même principe que celui utilisé lors de la connexion : un script d'envoi côté client en « **UnityScript** » dans la couche d'accès aux données et un script **PHP** « **AddScore.PHP** » hébergé en ligne sur le serveur. Celui-ci enregistre le score et sécurise les données reçues avec la fonction « **Protect()** » utilisée lors de la connexion ou de l'ajout d'un nouveau joueur.

III. Sprint 3 :

Création des menus et autres interfaces utilisateurs



1. Tableau prévisionnel du sprint

- *Date de début du sprint* : Mardi 1 Avril 2014
- *Date de fin de sprint* : Jeudi 16 Avril 2014
- *Temps estimé en heures* : 144 heures
- *Échelle de mesure* :
 - Une journée égale 8 heures de travail (de 9h->13h puis de 14h->18h)
 - On compte dans le tableau, les heures restantes en fin de journée (donc après 8 heures de travail).
- *Objectifs du sprint* :
 - Ajout des barres de vies et compteurs (joyaux et temps) sur la scène de jeu.
 - Ajout des pages de menu (connexion, choix personnage, rejoindre...)

Nom :	Priorité :	Temps :	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Conception des GUI	60	4 heures	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Peinture des GUI dans tous leurs états	61	8 heures	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Conception des pages du jeu	62	4 heures	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Conception des maquettes des pages	63	8 heures	8	8	0	0	0	0	0	0	0	0	0	0	0	0	0	
Conception du schéma de navigation entre les pages	64	4 heures	4	4	4	0	0	0	0	0	0	0	0	0	0	0	0	
Création des fonds et des logos	65	16 heures	1 6	1 6	1 6	1 6	8	0	0	0	0	0	0	0	0	0	0	

« Tableau 6 : première partie du mois - sprint 3 »

2. Création graphique des éléments d'interfaces

2.1. Cration des logos

L'application étant à présent jouable, il manquait une étape avant de considérer le jeu comme étant fini. En effet, tout jeu professionnel possède une interface permettant à l'utilisateur de comprendre et d'utiliser le jeu. Deux types d'interfaces sont pris en compte :

- Les indicateurs dans la partie qui permettront aux joueurs de connaître leurs niveaux de santé, le temps écoulé depuis le début de la partie ainsi que le nombre de joyaux récupéré.
 - Les composants des différents menus (boutons, champs de saisie, textes...).

Mais avant cela, nous avons choisi de commencer par créer le logo du jeu « The Twins » ainsi que celui de notre groupe « Prod'IT Studio ».

2.1.1. Cration du logo the Twins

Le logo du jeu devait à la fois refléter les deux personnages principaux, mais aussi leur environnement. Ainsi, nous avons référencé tous les éléments qui pourraient nous servir pour créer les formes ou les couleurs du logo :

- Les armes des deux personnages pour la forme du logo.
 - Les plantes, les vignes et les arbres pour la forme.
 - Les papillons volant librement dans le jeu.
 - Les ennemis du jeu pour la forme et la couleur.
 - Le ciel du jeu en volute pour la forme du logo.
 - Les joyaux, la balançoire et autres énigmes.

Avec ces éléments, nous avons finalement décidé que :

- Le « T » de Twins prendrait la forme de l'épée de « Almas ».
 - Le « i » de Twins prendrait la forme du sceptre de « Umi ».
 - Le point du « i » prendrait la forme d'un joyau.
 - Le « s » recevrait un papillon pour équilibrer les proportions du logo.

- Les autres lettres que le « T » et le « i » recevraient une police basée sur les volutes du ciel. Pour cette raison, nous nous sommes inspirés de la police « **Curlz MT** » et avons créé le logo.

La coloration des lettres (mis à part le joyau et le papillon) serait le jaune. Nous avons choisi cette couleur car avec le bleu du papillon et le rouge du joyau, le vert et le marron de « Almas » et du monde ne conviennent pas. De plus, choisir le vert met à l'écart « Umi » par rapport à « Almas ». Le jaune de plus rappelle la couleur de la peau et surtout des cheveux des deux personnages. Cette couleur permet donc de les inclure tous les deux. Enfin, nous n'avons consciemment pas mis de soleil dans le jeu. Nous avions considéré que les « Twins » représentaient la lumière du monde qu'ils sauvaient. Le jaune représentait donc la meilleure solution.



« Figure 130 : logo The Twins »

2.1.2. Création du logo de Prod'IT Studio

Les deux étudiants de notre groupe sont tous deux des membres d'une association tunisienne appelée « **Prod'IT** », Anas Neumann en est par ailleurs le co-fondateur et vice président. Cette association, fondée il y a maintenant deux ans par le réalisateur Ahmed Hermassi, est spécialisée dans la cinématographie, l'audio visuel et le multimédia. Elle a pour but la création d'échanges et de synergie entre les « **freelances** » tunisiens et les producteurs afin de créer de l'emploi et de développer ces arts dans le pays.

Souhaitant que notre groupe de travail et peut-être notre future société travaille en relation forte avec l'association « **Prod'IT** », nous avons décidé de l'appeler « **Prod'IT Studio** ».

À partir de cette base, il semblait évident que le logo du groupe soit lié à celui de « **Prod'IT** ». Ainsi nous l'avons pris comme référence de base et l'avons modifié de manière à ce qu'il se rapproche du monde du jeu vidéo.



« Figure 131 : logo Prod'it »

Nous avons donc repris les couleurs du logo : le noir, le blanc et le bleu turquoise. Ensuite, nous avons également repris la police et la forme générale de « bulle ». Par contre, la texture de la bulle étant liée au vieil écran de cinéma, elle ne correspondait pas.

Nous avons donc décidé d'ajouter de la 3D dans la police et la bulle ainsi que des effets de lumière pour rappeler le domaine « **High-Tech** » dont fait partie le jeu vidéo. Nous avons aussi choisi de superposer plusieurs bulles plutôt que d'en laisser qu'une seule. Ainsi, le sentiment d'un groupe uni sur une même idée serait plus fort.



« Figure 132 : Prod'it Studio »

2.2. Structure générale des pages

Pour faciliter la compréhension des menus et des pages par les enfants, nous avons décidé qu'elles suivraient toutes une même structure générale. Cependant, la scène de jeu ne peut quant à elle respecter cette organisation. Nous avons donc deux structures pour nos pages, une structure générale et une autre uniquement pour la scène de jeu.

2.1.1. Structure de la fenêtre de jeu

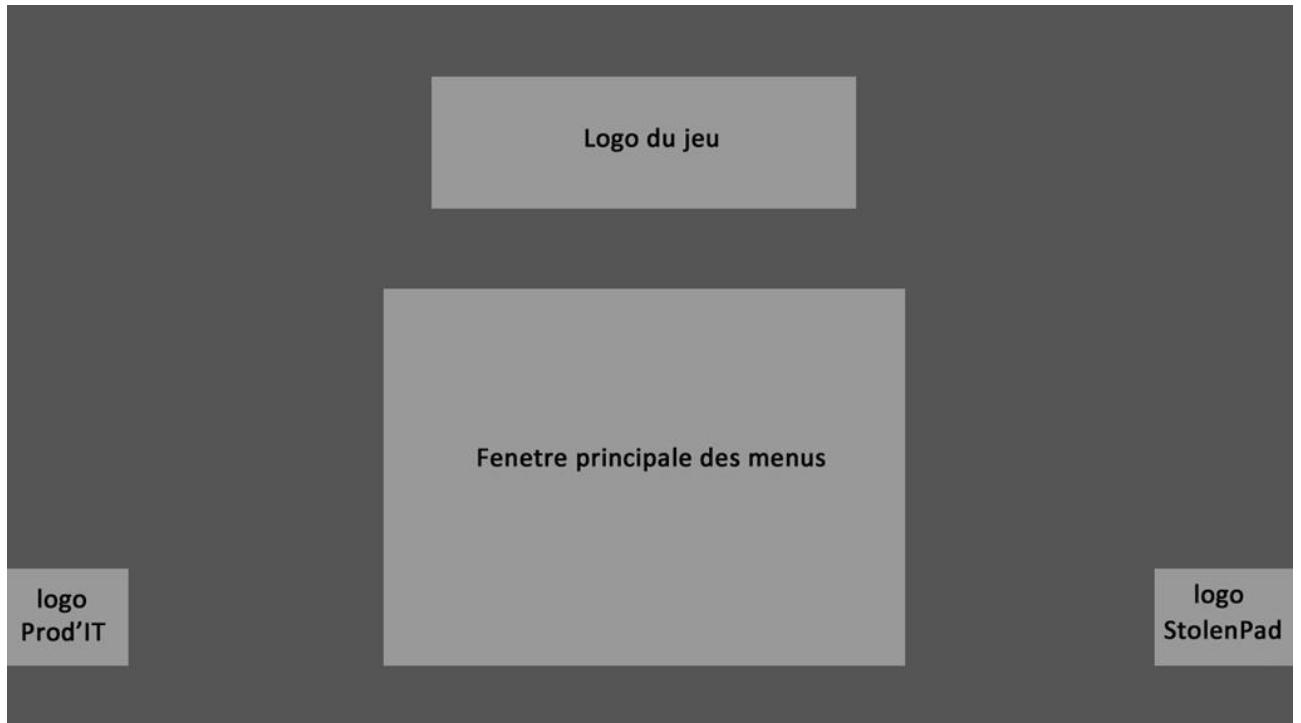
Les emplacements et les tailles ont été étudiés de façon à ce qu'ils ne gênent pas dans le jeu. En effet, Le personnage étant centré verticalement et horizontalement dans à l'écran, les angles sont les positions les moins utilisées dans le jeu. De plus, les ennemis et énigmes arrivant de la droite mais centrés verticalement, leurs arrivées ne seront pas cachées par les « GUI ».



« Figure 133 : structure spatiale de la fenêtre du jeu »

2.1.2. Structure générale des menus

Cette fois-ci au contraire, le joueur est concentré uniquement avec le menu qui doit donc être centré est prendre une partie conséquente de la page. Les logos des entreprises au contraire sont secondaires et sont donc moins exposés en avant.



« Figure 134 : structure spatiale des menus »

2.3. Création des éléments

Tout comme pour les structures, les éléments des menus et les GUI présents dans le jeu doivent être deux groupes différenciés. Cependant, nous avons gardé une cohérence de formes et de couleurs entre les deux groupes.

2.3.1. Les GUI du jeu

Les GUI se trouvant dans les angles inférieurs de l'écran (en bas), seraient superposés au dessus du décor. Le décor étant très coloré et assez chargé au niveau formes, les GUI seraient difficilement différenciables. Pour cette raison, nous avons décidé qu'ils auraient un fond noir. Cependant, pour ne pas cacher le décor, le noir utilisé aurait une opacité légèrement réduite. Ainsi, ils seraient suffisamment visibles mais ne cacheraient pas non plus le décor.

Pour la symbolique des GUI, nous avons décidé de faire simple et de garder à l'esprit que les joueurs seraient des enfants. Ainsi, le nombre de joyaux sera précédé d'une peinture d'un joyau et le temps écoulé sera précédé d'un sablier.

Le joyau étant rouge, nous avons choisi que l'écriture serait de la couleur complémentaire : le bleu. Cette couleur sera également utilisée pour le compteur du temps. Il faudra également que le bleu soit suffisamment clair pour être lisible sur le fond noir.

La police choisie est « **Curlz MT** », car nous nous sommes basés dessus pour créer celle du logo. De plus, cette police possède les formes de type « volute » que l'on trouve dans le ciel et les vignes du jeu.



« Figure 135 : GUI pour le compteur de joyaux »

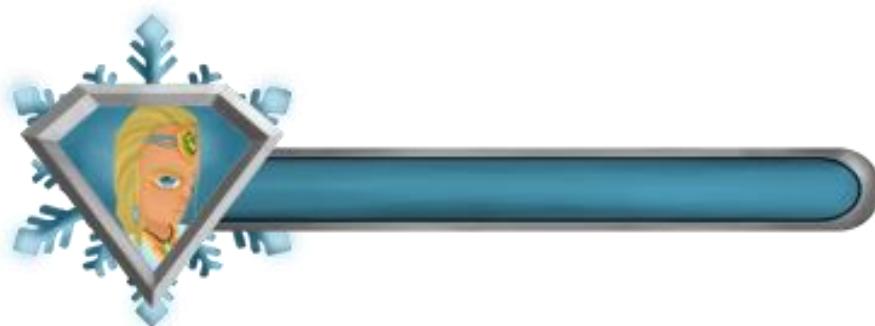


« Figure 136 : GUI pour le compteur de temps »

Les deux barres de vie se trouvant au dessus du ciel, elles seront plus facilement visibles. En effet, le ciel ne contient que deux couleurs (dont le blanc) et très peu de formes. Ainsi, nous avons pu nous permettre des GUI plus colorés et avec des formes plus recherchées.

Les barres de vie étant liées chacune à un personnage, nous avons décidé qu'elles seraient chacune représentative de ce personnage. Ainsi la barre de « Umi » possédera le thème du pouvoir de glace, du sceptre d'acier et les couleurs bleu et argent. La barre d'« Almas » sera composée du boulier, de l'épée ainsi que les couleurs rouge des joyaux et vert de ses vêtements.

Enfin, nous avons choisi que l'identification du personnage lié à la barre sera clarifiée pour les enfants par la présence de la tête du personnage.



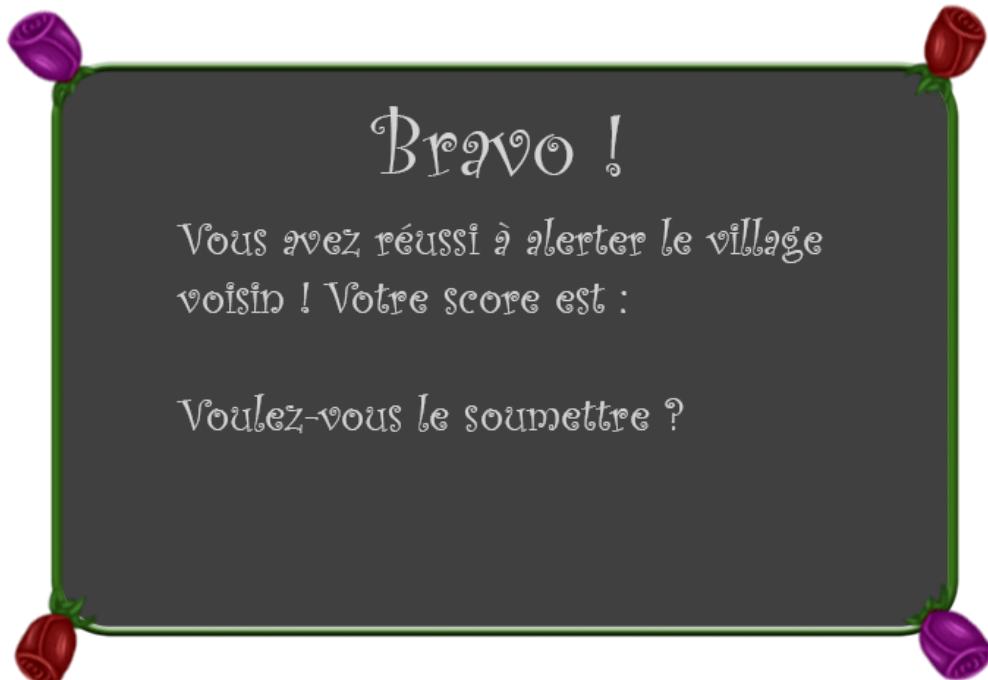
« Figure 137 : barre de santé - "Umi" »



« Figure 138 : barre de santé "Almas" »

2.3.2. Les éléments des menus

La fenêtre principale utilisée pour les menus subit les mêmes contraintes que les GUI des joyaux et du temps. Elle se trouve au dessus du monde 3D du jeu très chargé et très coloré. Ainsi nous avons décidé de reprendre l'idée du fond noir non opaque. Nous avons également repris la police « Curlz MT » mais en blanc cette fois-ci. Le blanc ne correspond à aucun des deux personnages et est la couleur la plus lisible sur un fond noir.



« Figure 139 : une des fenêtre du jeu »

Pour ajouter un peu de couleur, nous avons ajouté le contour vert rappelant l'univers forestier. Les boutons posséderont d'ailleurs cette même couleur. Ensuite, nous avons choisi que :

- Lors du passage de la souris, les boutons de retour ou d'annulation auront la couleur des roses rouges présentes aux angles de la fenêtre.
- Lors du passage de la souris, les boutons de confirmation ou de choix auront la couleur des roses violettes présentes également aux angles de la fenêtre.



« Figure 140 : bouton "Quitter" en surbrillance »



« Figure 141 : bouton "Quitter" »



« Figure 142 : bouton "Connexion" en surbrillance »



« Figure 143 : bouton "Connexion" »

2.4. L'interface complète

Pour finir, voici quelques exemples des menus et des interfaces une fois les éléments regroupés ensemble :



« Figure 144 : interface - scène du jeu »



« Figure 145 : interface - sélection de personnages »



« Figure 146 : interface - écran de connexion »

3. Développement des interfaces

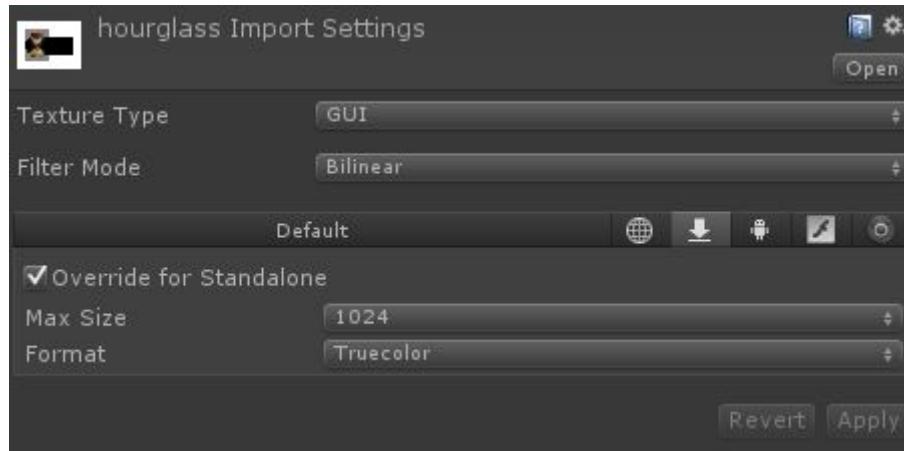
Pour que les interfaces utilisateurs soient entièrement fonctionnelles, leur développement dans « Unity » regroupe plusieurs étapes :

- L'importation des éléments (au format PNG si il y a de la transparence) avec des réglages spécifiques dans Unity.
- La création d'une classe qui effectuera le redimensionnement des boutons et autres GUI en fonction de la taille et des proportions de l'écran.
- La création des « components » de la couche « Présentation » qui seront utilisés pour passer d'une fenêtre à une autre. Les méthodes de ces classes appelleront en fait les méthodes des classes couche « Metier » pour par exemple démarrer une partie ou rejoindre un serveur.

3.1. Réglages des GUI Dans Unity

Pour éviter la compression et la perte de qualité, il faut spécifier lors de l'importation la nature de l'image. Pour cela on règle en tant que « GUI » affiché sans compression en « **True color** ».

Pour ne pas perdre de la mémoire, on spécifie aussi la taille maximale à accorder au fichier. Par exemple, si on importe un fichier de taille 512*512, on signale au moteur de ne pas gérer une taille plus importante que 512 pour ce fichier, cela est inutile.



« Figure 147 : importation d'image pour les GUI »

Après avoir bien importé les GUI, nous avions quatre solutions pour créer les menus et les autres GUI :

- Créer des plans en 3D et utiliser les éléments GUI comme menus. C'est la technique utilisée par l'entreprise qui nous encadre.
- Utiliser le « plug-in » « **Ngui** », très puissant et sophistiqué mais nous manquions de temps pour apprendre son utilisation.
- Utiliser la fonction de Unity « **OnGUI()** ». Cependant, les développeurs de **StolenPad Studio** nous l'on fortement déconseillée car elle est utilisée plus de mémoire que nécessaire.
- Utiliser les classes « **GUITexture** » et « **GUIText** » qui créent des GUI automatiquement mais ne gèrent pas nativement les proportions et tailles des écrans. Ces classes affichent les GUI avec leurs tailles à l'importation.

Parmi ces choix, nous avons choisi la dernière solution ce qui impliquait de créer un « component » spécialisé dans le redimensionnement et le repositionnement des GUI en fonction de la taille et des proportions de l'écran de l'utilisateur.

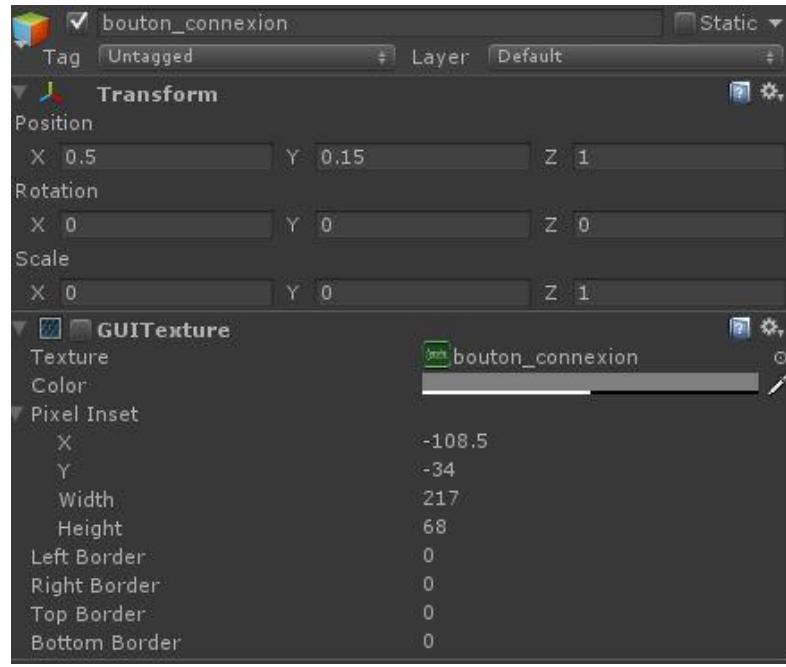
3.2. Un menu pour tous

Pour créer ce « component », la programmation se base sur les variables disponibles dans les classes « **GUIText** » et « **GUITexture** ».

3.2.1. Les variables **GUITexture**

Le positionnement d'un GUI de type image est différent de celle d'un objet 3D. Dans le cas d'un GUI, la position bas / gauche de l'écran correspond à $x=0$ et $y=0$ dans le « component » « **Transform** » et la position haut / droit à $x=1$ et $y=1$.

Ainsi, par exemple si l'on veut centrer un logo à l'écran on le positionne en $x=0.5$ et $y=0.5$. Cependant cela ne suffit pas. En effet la position d'un GUI ne dépend pas de son centre mais l'angle haut / gauche du celui-ci. Donc, si l'on veut réellement le centrer à l'écran, il faut créer un décalage de la moitié de sa hauteur dans la position en y et de la moitié de sa largeur dans la position en x . Pour cela on se sert de la variable « **Pixel Insert** » pour régler la taille du GUI et les décalages souhaités.



« Figure 148 : réglage GUI »

3.2.2. Les variables GUIText

Le positionnement du texte se fait de la même façon que les images cependant les réglages de la classe « **GUIText** » incluent des notions supplémentaires. Ces notions sont liées à l'affichage du texte comme par exemple la police, la taille ou la couleur. Pour ajouter des modifications supplémentaires de style au texte, il faut créer un « **GUISkin** » que l'on appliquera ensuite au texte avec de la programmation.

Tout comme pour les positions et tailles des images, les « **GUIText** » ont les problèmes que la taille du texte et sa position ne s'adaptent pas naturellement aux différences d'écran.



« Figure 149 : réglage GUIText »

3.2.3. La classe « GUITaille »

Nous avons enfin créé le « component » qui corrige ces problèmes, la classe « **GUITaille** » dont chaque GUI possédera une instance.

Cette classe modifie les valeurs des variables contenues dans « Pixel Insert » de la classe « **GUITexture** » en utilisant les variables « **Sreen.width** » et « **Sreen.height** » de la classe « **Screen** » pour connaître la taille et la proportion hauteur / largeur de l'écran de jeu.

Pour cela, nous avons imaginé le principe suivant :

- Les GUI sont créés et positionnés en fonction de la taille de notre écran
- Lors du démarrage du jeu, on vérifie si l'écran du joueur possède les mêmes proportions, possède un **rapport hauteur / largeur** plus faible ou encore un rapport plus fort.
- Si le **rapport hauteur / largeur** est plus faible, on change la taille et la position du GUI en fonction de la **largeur** de l'écran utilisé.
- Si le **rapport hauteur / largeur** est plus fort, on change la taille et la position du GUI en fonction de la **hauteur** de l'écran utilisé.
- Dans le cas d'un texte, on vérifie également dans un autre script la **grandeur** de cette différence de proportion pour la compenser en décalage du « **GUIText** ».

3.2.4. Un passage tout en douceur

Cette classe, bien qu'elle ne soit pas la seule dont nous nous servons pour cela, possède également une fonction d'apparition non instantanée du GUI. On entend par « non instantanée » le fait que le GUI apparaît avec une opacité réduite mais qui augmente petit à petit pour finir par être complète. Ainsi lorsque l'on passe d'un menu à un autre, le changement n'est pas brusque mais semblerait au contraire doux à l'œil de l'utilisateur.

Pour réaliser techniquement cela, nous modifions la valeur de la variable « **a** » contenue dans la variable « **color** » du « component » « **GuiTexture** ». La valeur de cette variable est stockée dans l'une de nos variables appelée « **alpha** » puis prend la valeur zéro. Par la suite, à chaque calcul d'une nouvelle image, la variable « **a** » est incrémentée tant qu'elle n'a pas retrouvé la valeur « **alpha** ».

3.2. Développement des fonctionnalités des menus et des GUI

3.2.1. Passage entre les pages

Le passage entre les différents menus du jeu correspond techniquement à un passage d'une scène Unity à une autre. Pour cela, nous avons créé le « component » « **BoutonControl** » qui prend dans une variable de type « String » le nom de la scène à charger lorsqu'on appuie sur le bouton. Nous utilisons la fonction « **LoadLevel()** » de la classe « **Application** » pour charger la scène et la fonction « **OnMouseUp()** » héritée de « **MonoBehaviour** » pour détecter l'appui de la souris sur le bouton.

Dans certaines scènes, le passage d'une page à l'autre est automatique. C'est le cas des scènes dans lesquelles nous présentons les différents logos (StolenPad Studio, Prod'IT Studio et The Twins). Dans ces scènes, on crée un système de compteur qui déclenche la fonction « **LoadLevel()** » après avoir atteint une certaine valeur.

3.2.2. Changement de texture

Le changement de texture lors du passage de la souris se fait avec une modification de la valeur de la variable « **texture** » du « component » « **GUITexture** ». La vérification du passage de la souris se fait avec des fonctions héritées de la classe « **MonoBehaviour** ». Ces fonctions sont : « **OnMouseEnter()** » et « **OnMouseExit()** ».

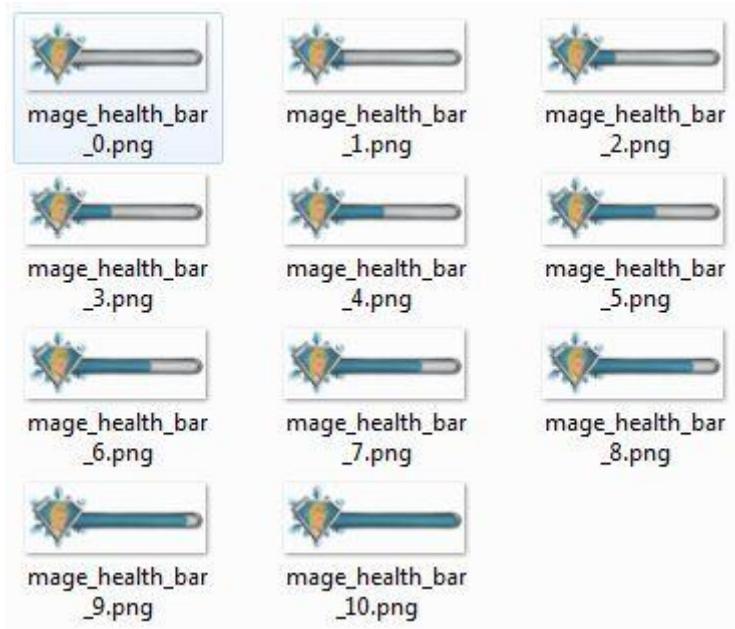
3.2.3. Les compteurs des joyaux et du temps

Les GUI qui affichent les valeurs des joyaux et du temps affichent en fait un simple message stocké dans une variable de type « **String** ». Le message n'est créé que sur l'ordinateur d'un seul des deux joueurs et est envoyé en RPC à l'autre. Par exemple le temps est calculé par l'ordinateur du joueur **client** qui l'envoie à celui du joueur **serveur**. Le nombre de joyaux quant à lui, est calculé par l'ordinateur du joueur qui contrôle « Almas » et qui l'envoie au joueur « Umi ».

3.2.4. Les barres de vie des personnages

C'est l'ordinateur du joueur concerné par la barre qui calcule la valeur de la vie du personnage et l'envoie en réseau. À partir de la valeur de la variable **vie**, chaque ordinateur calcule séparément quelle est la texture que la barre doit avoir. Une barre de vie contient toutes les textures nécessaires pour toutes les valeurs de la vie dans une variable de type « **Texture2D[]** ».

De cette manière, on limite la taille de la variable en réseau qui aurait été trop importante si on aurait envoyé le changement de texture. De plus, l'utilisation d'un tableau plutôt que d'une vérification à l'aide de « **if** » et « **else** » est plus optimisée en termes de calcul.





L'aventure
s'achève

IV. Sprint 4 :

Création du site web et des autres supports



1. Tableau prévisionnel du sprint

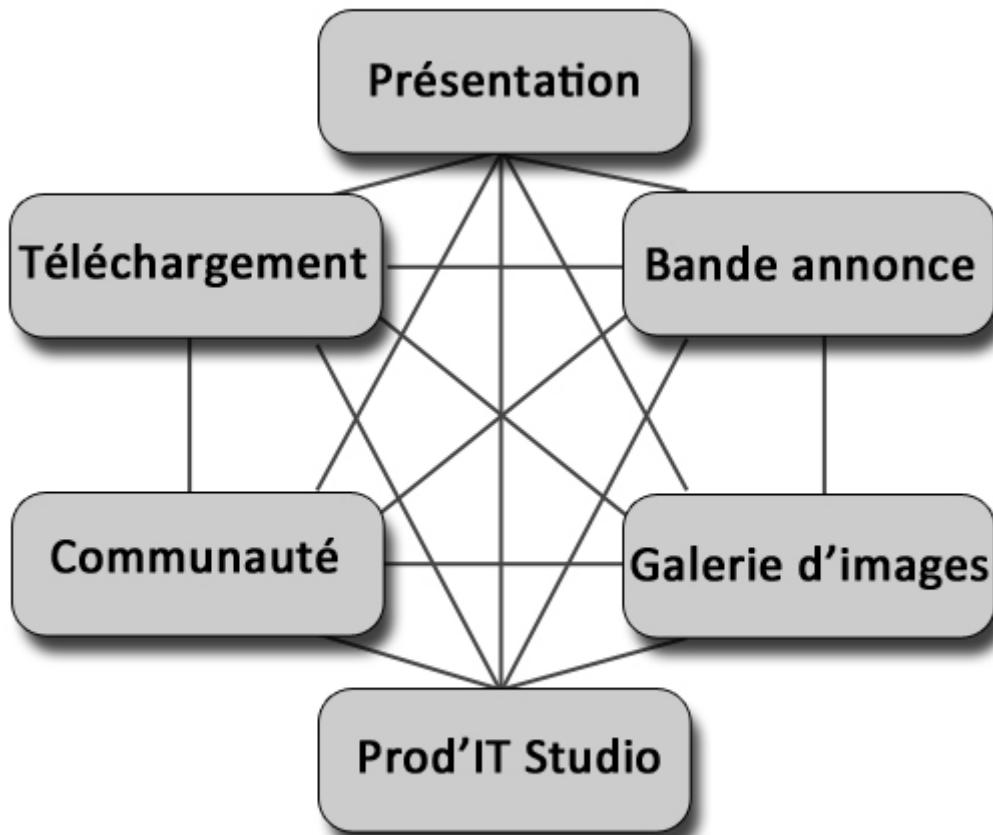
- *Date de début du sprint :* Jeudi 17 Avril 2014.
- *Date de fin de sprint :* Mercredi 30 Avril 2014.
- *Temps estimé en heures :* 76 heures.
- *Échelle de mesure :*
 - Une journée égale 8 heures de travail (de 9h->13h puis de 14h->18h).
 - On compte dans le tableau, les heures restantes en fin de journée (donc après 8 heures de travail).
- *Objectifs du sprint :*
 - *Création du site web sur lequel les internautes pourront découvrir et télécharger notre jeu.*
 - *Création d'une description sur le site pour le groupe « Prod'IT Studio ».*
 - *Création d'autres supports tels qu'une page Facebook ou encore une chaîne Youtube.*

Nom	Priorité	Temps	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Choix des formes et des couleurs du site web	77	16h	8	0	0	0	0	0	0	0	0	0	0	0	0	0
Conception schéma de navigation du site web	79	4h	4	4	0	0	0	0	0	0	0	0	0	0	0	0
Développement HMTL	80	16h	16	16	12	4	0	0	0	0	0	0	0	0	0	0
Intégration CSS et JavaScript	81	16h	16	16	16	12	4	0	0	0	0	0	0	0	0	0
Développement page PHP d'inscription	82	4h	4	4	4	4	4	0	0	0	0	0	0	0	0	0
Rédaction du contenu	83	8h	8	8	8	8	8	8	0	0	0	0	0	0	0	0
Intégration d'une vidéo du jeu	84	4h	4	4	4	4	4	4	4	0	0	0	0	0	0	0
Création de la page Facebook, Google+ et Youtube de Prod'IT Studio	85	4h	4	4	4	4	4	4	4	0	0	0	0	0	0	0
Hébergement en ligne du site web	86	4h	4	4	4	4	4	4	4	4	0	0	0	0	0	0

« Tableau 7 : deuxième partie du mois - sprint 4 »

2. Présentation du site web

Comme nous l'avons présenté dans sa conception, le site web sera principalement utilisé pour découvrir et télécharger le jeu. Pour faciliter son utilisation, il n'y aura pas de sous-menu et il ne contiendra donc qu'un nombre réduit de pages. Le **schéma de navigation** sera **étoilé**. Pour cela, un même menu sera présent dans toutes les pages et permettra de naviguer entre elles.



« Figure 151 : schéma de navigation étoilé »

Nous avons également décidé de séparer les dossiers en deux : un dossier « **html** » contenant la couche « présentation » exécutée chez le client et un dossier « **php** » contenant la couche « métier » et « accès aux données » exécutées sur le serveur.

Cependant, pour être complet, le contenu du dossier « **html** » utilise le contenu de deux autres dossiers : « **images** » et « **css** ».

3. Crédit graphique

3.1. Crédit des éléments des pages

Nous avons commencé par sélectionner les éléments du jeu qui pourraient être utilisés dans le site web. En effet, nous souhaitions que le design du site web soit lié au jeu. Ainsi, nous avons choisi que :

- Le fond du site serait le ciel du jeu.



« Figure 152 : ciel du jeu »

- Les puces seront des joyaux ressemblant à ceux du jeu.



« Figure 153 : puce 1 »



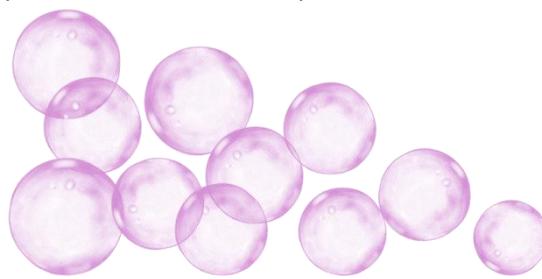
« Figure 154 : puce 2 »

- L'icône du site affichée en haut du navigateur sera les armes.



« Figure 155 : armes »

- Le haut de la page possédera les bulles des plantes et des lacs empoisonnés.



« Figure 156 : bulles »

- Nous placerons derrière le texte des fonds contenant des silhouettes d'éléments du jeu tels que celui-ci.



« Figure 157 : silhouettes »

3.1 Structure des pages du site

Pour faciliter la compréhension des utilisateurs, toutes les pages suivent la même structure. Nous voulons aussi éviter le défilement avec la mollette de la souris. En effet, il est toujours plus facile d'avoir toutes les informations présentes devant soit.

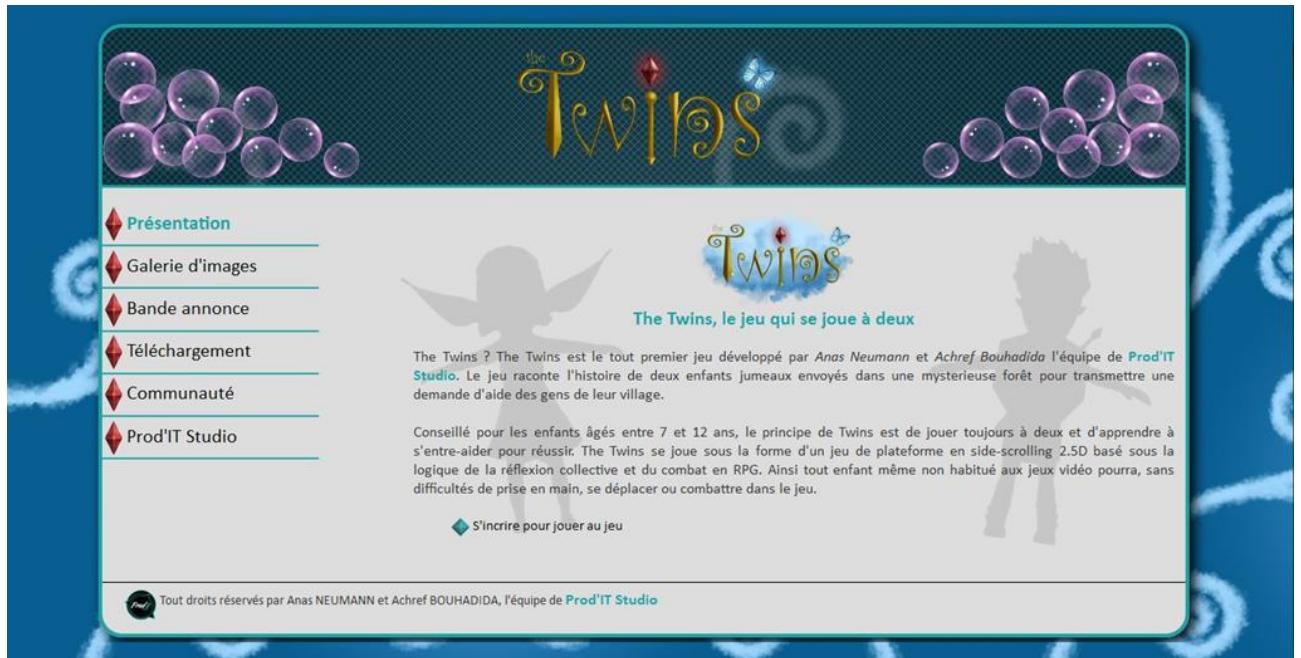


« Figure 158 : structure spatiale du site »

Les menus à gauche permettent la navigation en réseau que nous souhaitions. Pour supprimer l'ascenseur de défilement, nous avons donné au centre de la fenêtre la forme d'une application.

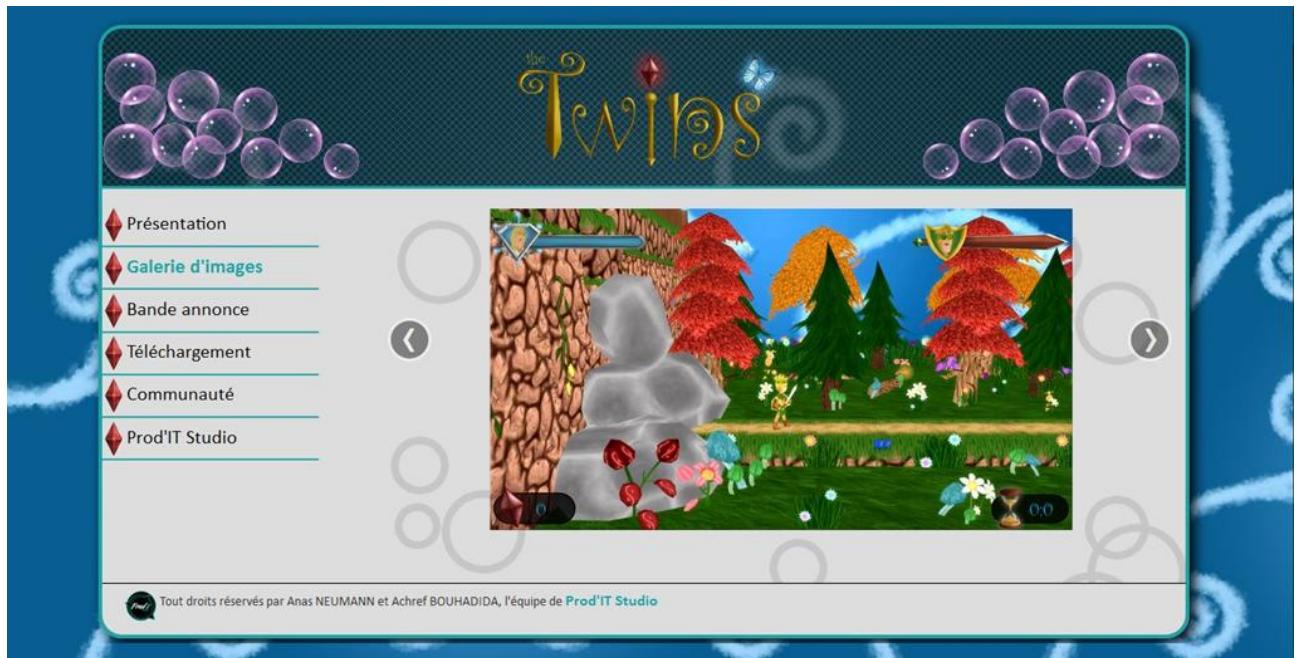
Voici les différentes pages une fois les éléments intégrés :

- La page d'index qui présente le concept du jeu de manière à éveiller la curiosité. On peut également trouver sur cette page un lien vers l'inscription.



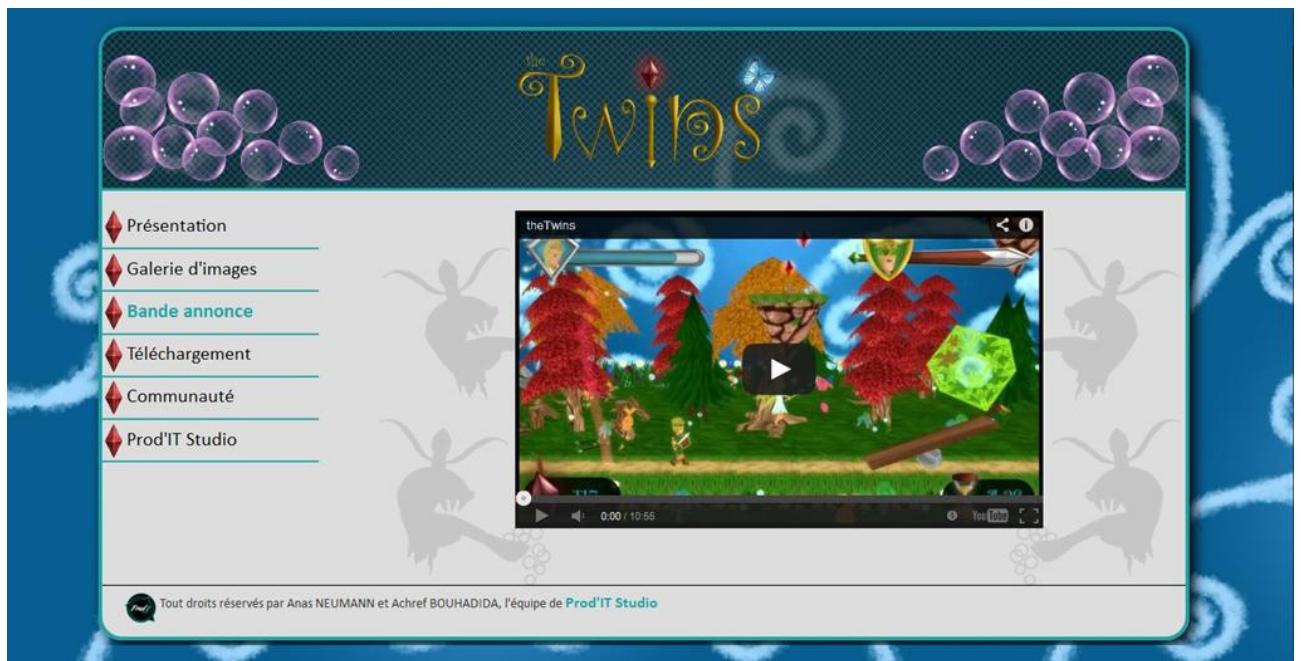
« Figure 159 : page index du site »

- La page « Galerie d'images » qui propose aux internautes de visualiser des captures d'écran du jeu afin qu'ils puissent se faire une idée sur les graphismes du jeu.



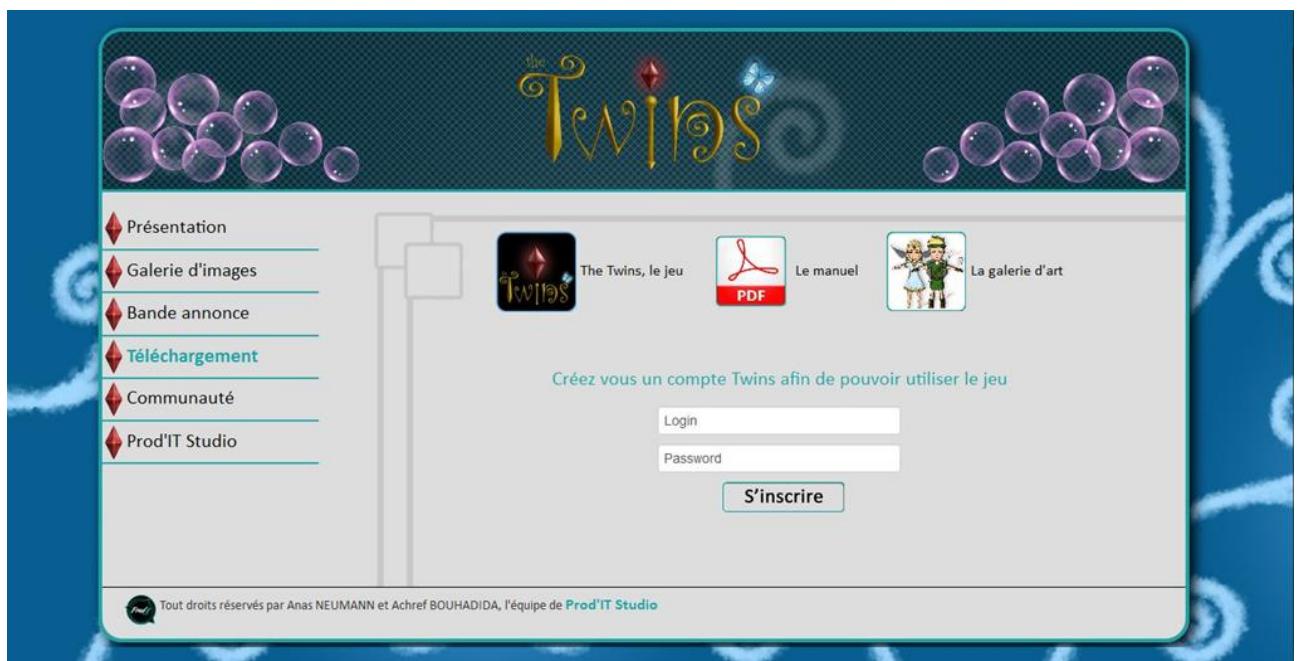
« Figure 160 : page "Galerie d'images" du site »

- La page « Bande annonce » dans laquelle est intégrée une vidéo hébergée sur « Youtube ». Cette vidéo permet aux joueurs d'en apprendre davantage sur le « gameplay » et l'utilisation du jeu.



« Figure 161 : page "Bande annonce" du site »

- La page de téléchargement sur laquelle on peut télécharger le jeu, le manuel d'utilisation ainsi que la galerie d'art pour les fans. On peut également s'inscrire sur cette page.



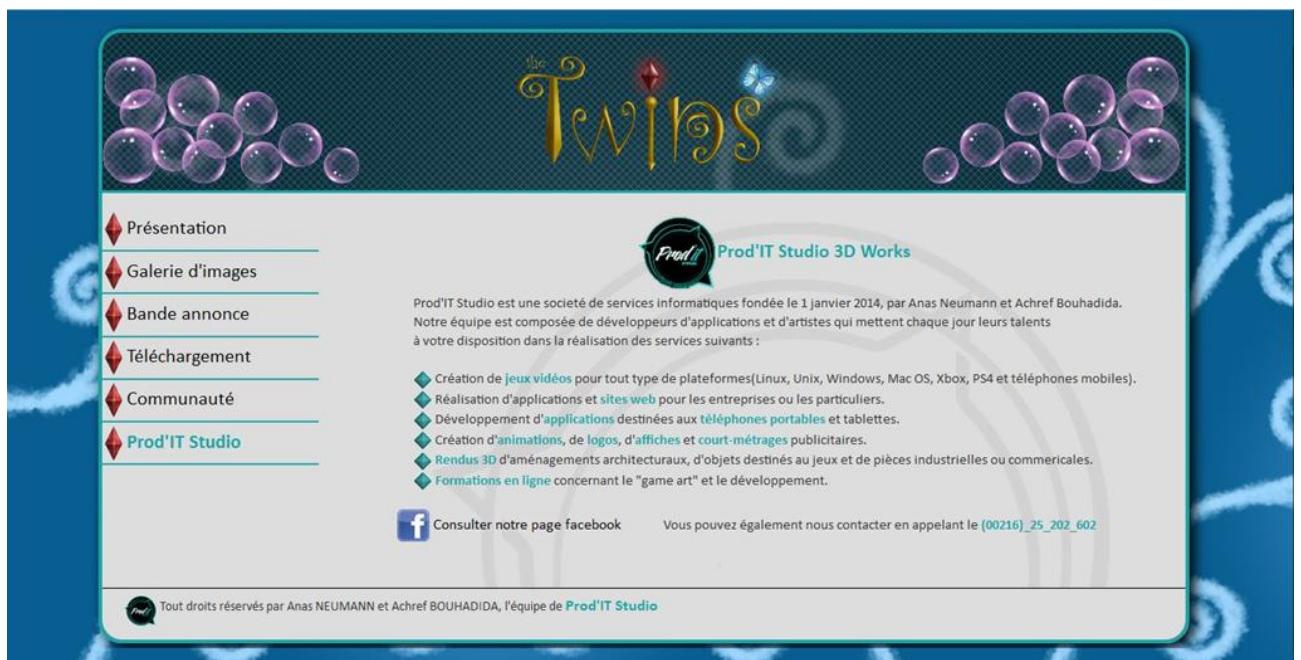
« Figure 162 : page "Téléchargement" du site »

- La page « Communauté » dans laquelle chaque internaute pourra visualiser les meilleurs scores fait par les joueurs ainsi que les trophées reçus. Le meilleur score reçoit une coupe d'or, le deuxième reçoit une coupe d'argent et le troisième une coupe de bronze.



« Figure 163 : page "Communauté" du site »

- Enfin, la page Prod'IT Studio permet aux internautes de découvrir la société.



« Figure 164 : page "Prod'IT Studio" du site »

4. Développement des pages du site

4.1. Sécurité des dossiers

Notre site web ne possédant pas de mécanisme permettant de cacher l'**URL**, il faut mettre au point un processus de protection contre l'entrée non désirée dans les dossiers.

Par exemple, lorsqu'un internaute se trouve sur la page d'accueil de notre site, il voit l'URL suivante sur son navigateur « `http://www.proditstudio.abdev.fr/html/index.html` ». S'il supprime la fin de l'URL et qu'il entre donc « `http://www.proditstudio.abdev.fr/html/` », il sera redirigé automatiquement vers notre page d'index et ne pourra donc pas visualiser le contenu du dossier « hmtl ». Cette redirection automatique est due au fait que le dossier contient un fichier nommé « `index.*` ». Si un internaute entre donc l'URL d'un autre dossier tel que « `http://www.proditstudio.abdev.fr/css/` » qui ne contient pas la page d'index, il pourra visualiser le contenu du dossier.

Pour remédier à ce problème, il suffit donc de placer dans tous les dossiers et sous-dossiers du site web un fichier appelé « `index.php` » qui effectue tout simplement une redirection directe vers la véritable page d'index à l'aide de la fonction « `Header()` ». Ainsi quelque soit l'URL testée par l'utilisateur, il se retrouvera toujours sur la page qui lui est destinée.

4.2. Développement HTML/CSS

Après avoir sécurisé nos dossiers, nous avons commencé à développer les pages statiques du site web. Comme pour les scripts du jeu, les documents HMTL respectent tous une même structure pour simplifier la lecture et la possible maintenance ou évolution. Voici la structure générale et les éléments qu'elle contient :

```
<hmtl>
  <head>
    <!--
      Dans cette zone on spécifie :
      - Le nom du site, l'icône pour le navigateur,
      - Les feuilles de style css utilisées, une développée par nous et une
        pour les objets bootstrap,
      - Le script bootstrap de type « javascript » utilisé.

    -->
  </head>
  <body>
    <div class="centrePage">
      <!--
        Cette zone possède le fond « ciel bleu » qui s'adapte à la taille de
        l'écran grâce à la l'instruction css « background-size : cover ; ».

      -->
    <div class="fenetrePrincipal">
      <!--
        Cette zone possède deux effets css particuliers : elle émet de l'ombre
        et possède un contour arrondi. Pour cela on utilise les syntaxes css «
        box-shadow » et « border-radius ». Elle contient les deux zones qui
        vont suivre : les menus à gauche et la zone principale.

      →
    <div class="menuGauche">
      <!--
        Cette zone contient le menu de navigation. Il est composé de
        lien hypertexte utilisant la balise hmtl « <a></a> ». On peut
```

```

        aussi y trouver les images des joyaux qui s'adaptent à
        l'écran car leur taille est calculée en pourcentage.

        -->
    </div>
    <div class="partieCentrale">
        <!--
            Cette zone est la seule à changer d'une page à une autre. On
            y trouve le sujet principal de la page.
        -->
    </div>
</div>
<!--
    Ici une bordure noire créée avec un bloc très fin de couleur noir.
    Puis, en dessous, un lien hypertexte vers la page facebook de
    Prod'IT Studio.
-->
</div>
</body>
<html>

```

« Figure 165 : structure des documents HTML »

Il faut également savoir que :

- Le téléchargement se fait simplement à l'aide d'un lien hypertexte vers la position du fichier à télécharger.
- Les tailles sont calculées en pourcentages afin d'avoir un site web qui s'adapte avec toutes les proportions et tailles d'écrans réglementés. Le site peut également s'adapter lorsque l'utilisateur déforme la taille et les proportions de sa fenêtre.

4.2. L'outil « bootstrap »

Pour créer la page « Galerie d'images » et pouvoir y placer un certain nombre d'images sans que l'utilisateur soit obligé de défiler verticalement entre elles, nous avons intégré du « **bootstrap** ».

Disponible à l'adresse « <http://getbootstrap.com/2.3.2/javascript.html> », « **bootstrap** » est un « FrameWork » facilitant la création de site web. En effet, télécharger « **bootstrap** » signifie en fait télécharger un fichier **javascript** pour les animations, un fichier **css** pour la mise en page des éléments et une banque d'images. Après quoi, il faut appeler ces deux fichiers au début de la page html pour ensuite pouvoir utiliser des éléments bootstrap tels que des boutons, fenêtres superposées, champs animés, etc...

De tous ces éléments disponibles via bootstrap, nous avons utilisé celui qui permet de faire dérouler des images automatiquement comme un diaporama. Cet élément s'appelle le « **carousel** ». Cet élément est en fait tout simplement une balise « **div** » signalée en tant que « **carousel** » via sa « **class** » et son « **id** ». Dans ce bloc, on place les images dans des sous-blocs ayant comme « **class** », la classe « **item** » pour signaler que l'image est une « **slide** » du « **carousel** ». Enfin, deux boutons permettent d'avancer manuellement d'une image à une autre. Pour ce faire, on crée deux liens hypertextes dont la référence est une ancre placée sur le **carousel** de la manière suivante : «

`href="#MyCarousel" » et l'indicateur pour avancer et reculer interagissant avec le javascript est spécifié comme suit : « data-slide="prev" ».`

4.3. L'inscription et les meilleurs scores

4.3.1. Les meilleures scores

La page des meilleurs scores en ligne est la seule page dynamique du site. La partie centrale n'est pas un texte figé car il doit s'adapter à chaque nouveau meilleur score fait. Ainsi, on utilise le langage PHP pour interagir avec la base de données et récupérer les meilleurs scores.

On utilise PHP dans sa version 5 pour créer des objets instances de classes « **entités** » et on effectue les vérifications complexes sur les objets plutôt que directement dans la base de données.

Pour ces vérifications complexes, on utilise la méthode la plus classique en PHP. On enregistre toutes les valeurs de chaque table de la base de données dans des objets. Ces objets sont eux-mêmes enregistrés dans des listes. Il y a une liste pour les utilisateurs, les parties et les scores. Ensuite, on parcourt ces listes avec la fonction php « **foreach()** ». On imbrique les boucles les unes dans les autres pour obtenir des vérifications multiples.

4.3.2. L'inscription

La page de téléchargement, bien que statique, contient un formulaire d'inscription qui envoie les champs à un fichier de traitement php : le fichier « **AddUser.php** ». Ce fichier effectue les étapes conçues dans le diagramme de séquences :

- Il sécurise les données reçues au cas où elles contiennent des tentatives d'injection **javascript**, **php** ou encore **sql**. Par exemple, si un utilisateur entre comme pseudo « **Sharley";DELETE FROM `user` WHERE `pseudo`="sharley";** », cet instruction sql ne sera pas exécutée mais lué comme une simple chaîne de caractères. Pour cela nous avons créé la fonction php suivante :

```
public static function protect($arg)
{
    $conn = mysqli_connect(SERVER, USER, PASSWORD, DB_NAME);
    $return=mysqli_real_escape_string($conn, strip_tags($arg));
    mysqli_close($conn);
    return $return;
}
```

« Figure 166 : fonction contre l'injection »

Cette fonction est tout simplement l'encapsulation d'une fonction php dans une autre. La première supprime tous les octets nuls et balises, et la deuxième supprime les sorties de chaînes de caractères et autres caractères spéciaux.

- Il vérifie que la valeur de la variable « **pseudo** », à présent sécurisée, n'existe pas dans la base de données avec l'instruction « **Select** ». Si le pseudo existe déjà, l'inscription ne sera pas faite et l'internaute sera redirigé vers la page « **erreur.html** ».



« Figure 167 : page d'erreur »

- Si le pseudo n'existe pas, on enregistre le nouvel utilisateur mais avant ça, on chiffre le mot de passe. Pour cela, nous avons créé une version PHP de l'**« algorithme de Jules César »** avec une clé de chiffrement égale à 4. Cette fonction traite aussi les chiffres mais par inversion.

```
public static function encrypte($document)
{
    /* La liste des substitutions par décalage de jules césar (lettres minuscules) */
    $juleCesar =
    array("a"=>"e","b"=>"f","c"=>"g","d"=>"h","e"=>"i","f"=>"j","g"=>"k","h"=>"l","i"=>"m","j"=>"n","k"=
    >"o","l"=>"p","m"=>"q","n"=>"r","o"=>"s","p"=>"t","q"=>"u","r"=>"v","s"=>"w","t"=>"x","u"=>"y","v"
    =>"z","w"=>"a","x"=>"b","y"=>"c","z"=>"d");

    $cryptogramme = strtr($document,$juleCesar); //la chaine après Jules César.

    /* La liste des substitutions par décalage de jules césar (lettres majuscules) */
    $juleCesarMaj =
    array("A"=>"E","B"=>"F","C"=>"G","D"=>"H","E"=>"I","F"=>"J","G"=>"K","H"=>"L","I"=>"M","J"=>"N",
    "K"=>"O","L"=>"P","M"=>"Q","N"=>"R","O"=>"S","P"=>"T","Q"=>"U","R"=>"V","S"=>"W","T"=>"X",
    "U"=>"Y","V"=>"Z","W"=>"A","X"=>"B","Y"=>"C","Z"=>"D");

    $cryptogramme = strtr($cryptogramme,$juleCesarMaj); //la chaine après Jules César sur les
    majuscules.

    /* La liste des inversions des chiffre avec substitutions */
    $inversion =
    ("1"=>"0","2"=>"9","3"=>"8","4"=>"7","5"=>"6","6"=>"5","7"=>"4","8"=>"3","9"=>"2","0"=>"1");
    $cryptogramme = strtr($cryptogramme,$inversion); //la chaine après l'inversion des chiffres.
```

```
//on laisse les caractères spéciaux intacts.
```

```
return $cryptogramme;
```

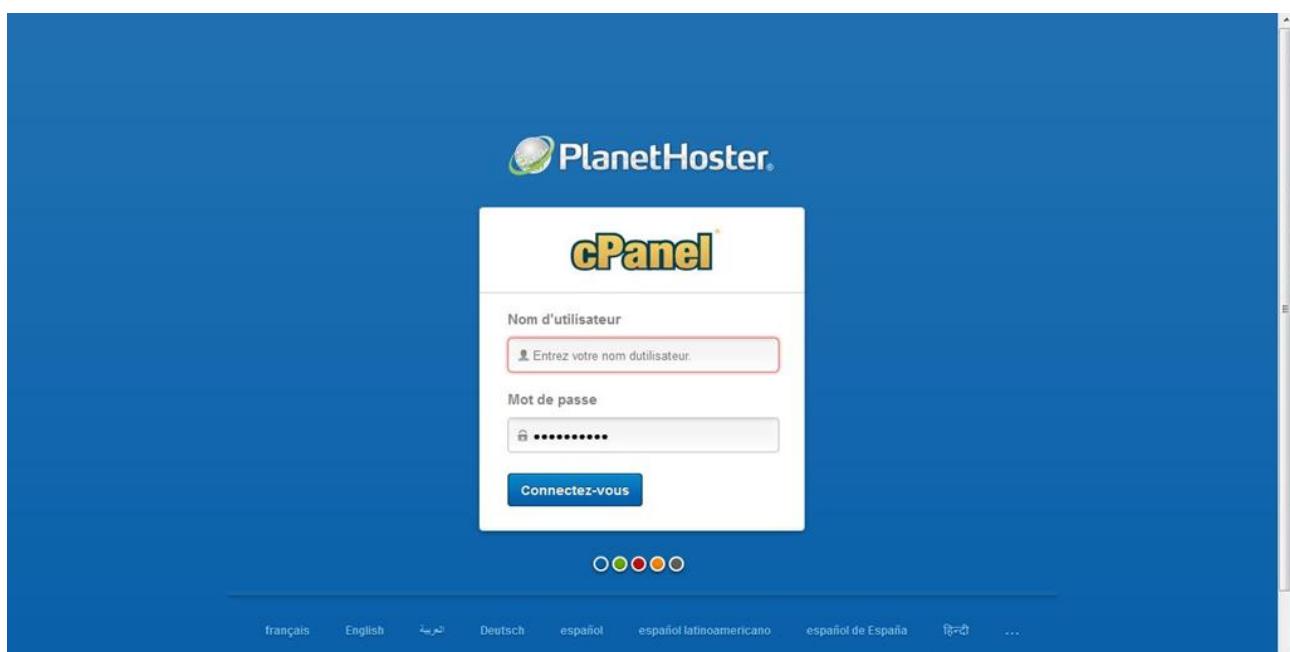
```
}
```

« Figure 168 : Jules Cesar »

- Une fois que l'inscription est terminée, l'internaute est redirigé vers la page « **bravo.html** ».

5. Hébergement du site web

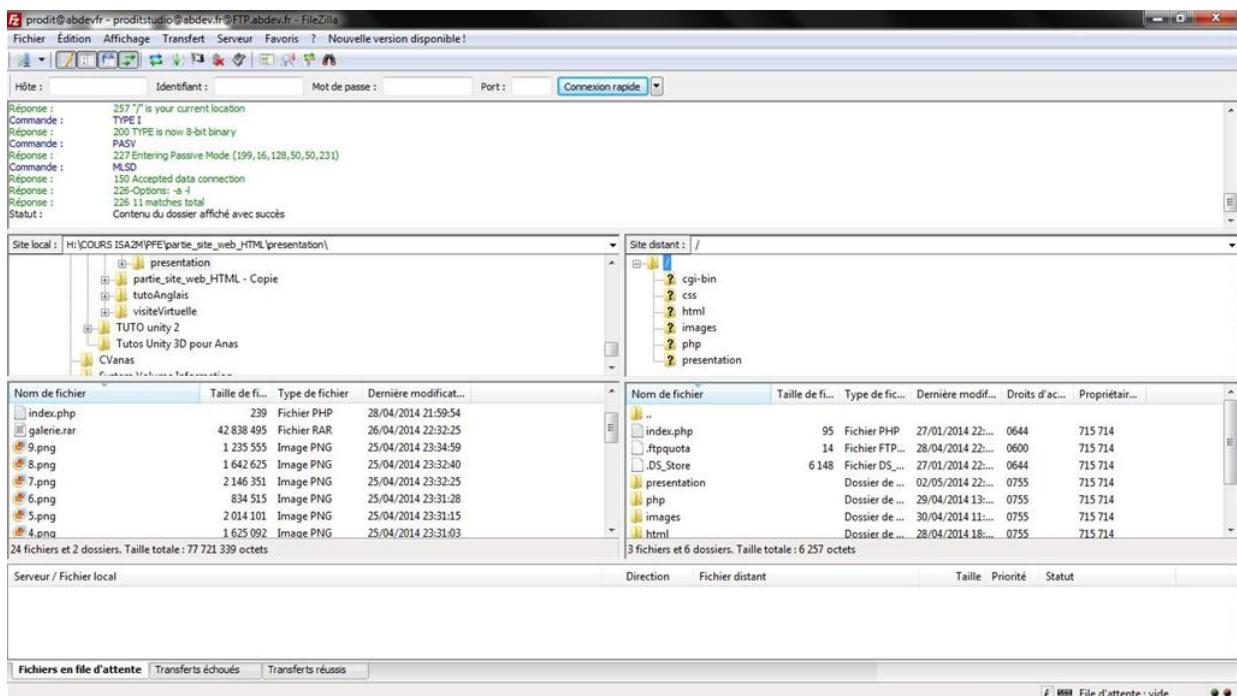
Pour que le jeu puisse être fonctionnel sur tous les ordinateurs et que le site puisse être vu de tous, il faut héberger sur un serveur la base de données ainsi que les pages du site. Nous avons donc tout hébergé sur un serveur payant de la compagnie « **Planet Hoster** ».



« Figure 169 : cPanel – PlanetHoster »

Pour ajouter le site web, il faut utiliser l'outil en ligne « **cPanel** » de « **Planet Hoster** ». Nous possédions déjà un compte et un nom de domaine « **abdev.fr** » et le type de compte que nous possédions permet l'ajout sans limite de sous-noms de domaine et donc de site web. Ainsi nous avons pu ajouter le site en tant que « <http://www.proditstudio.abdev.fr> » en suivant les étapes suivantes :

- Création du **domaine** que nous venons de mentionner.
- Création du **répertoire** dans lesquel les fichiers du site web seront enregistrés.
- Création du **FTP**, l'envoi des fichiers dans le répertoire se fait via un **FTP** qui sera utilisé dans le logiciel « **FileZilla** ».



« Figure 170 : FileZilla »

- Création de la base de données et des trois tables en **sql**. Pour cela, le « **cPanel** » contient une version de « **PHPMyAdmin** ». C'est un outil très performant notamment pour la gestion d'une base de données « **MySQL** ».
- Création d'un nouvel **utilisateur**. Cette étape n'est pas obligatoire mais il est tout de même préférable d'avoir un compte utilisateur pour chaque nouveau site web. De plus le « **cPanel** » permet l'ajout d'utilisateurs restreints.
- Si on a crée un utilisateur, il faut le lier avec la base de données que l'on a créée pour qu'il puisse la gérer.

6. Plusieurs supports de distribution

De nos jours, un site web n'est plus suffisant pour faire connaître son produit. Il est très important d'être également très présent sur les réseaux sociaux. Nous avons donc choisi de renforcer la présence de The Twins en l'intégrant dans les deux réseaux sociaux les plus utilisés ces dernières années : « **Youtube** » et « **Facebook** ».

6.1. La chaîne Youtube

Pour intégrer la vidéo dans notre site sans avoir à l'héberger de nous-mêmes, il fallait l'héberger sur un serveur spécialisé dans les vidéos. Youtube est le réseau de partage de vidéos le plus utilisé au monde et il nous permet facilement de partager par la suite la vidéo sur d'autres réseaux tels que facebook, par exemple.

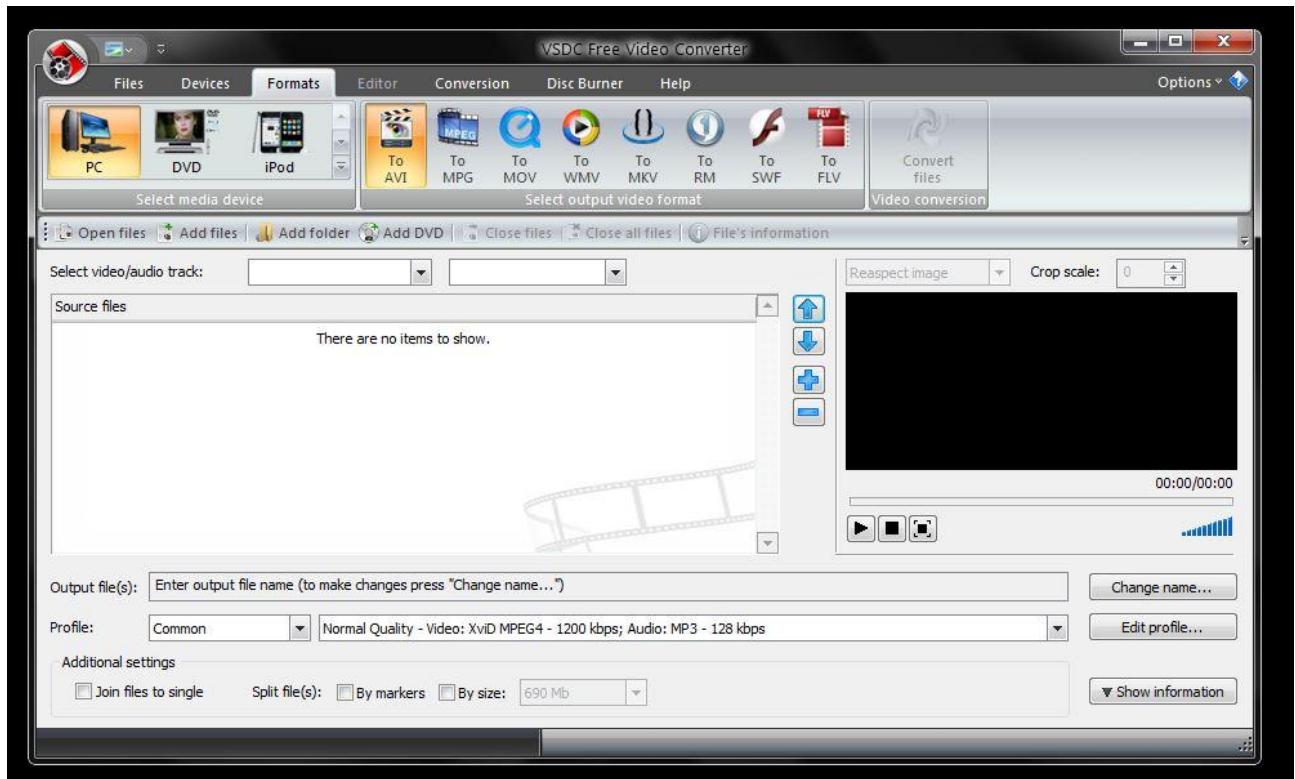
Pour cela il a fallu passer par quatre étapes :

- L'enregistrement de la vidéo. Pour cela, nous avons capturé l'écran de jeu durant tout le long d'une partie avec un logiciel appelé « **Fraps** ». Le logiciel permet de garantir une très bonne qualité mais produit en contrepartie une vidéo de grande taille en termes d'octets.



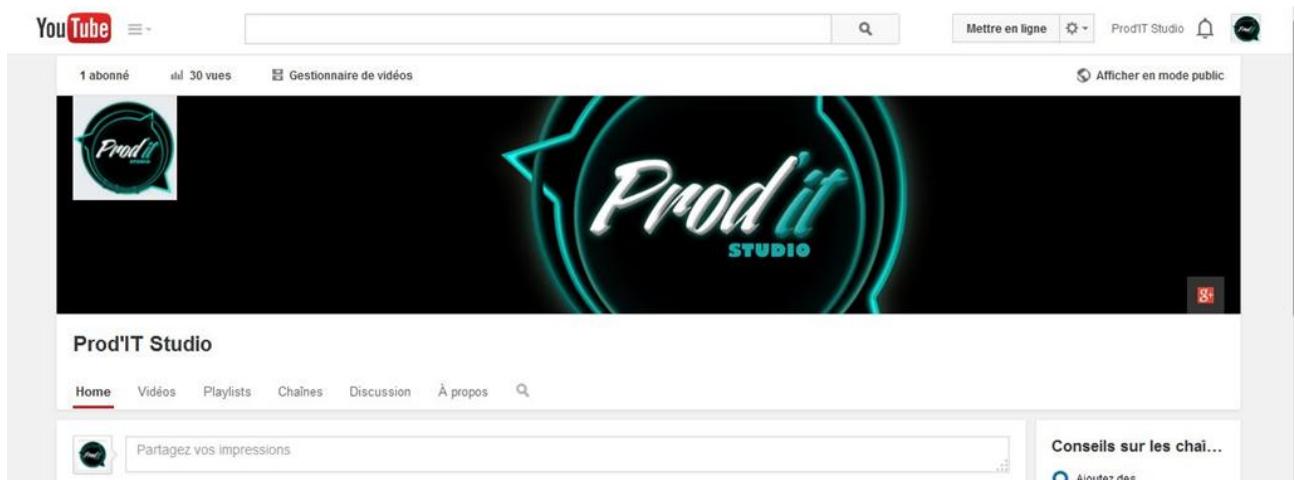
« Figure 171 : Fraps »

- Réduire la taille de la vidéo avec un logiciel gratuit de compression appelé « **VSDC** ». La vidéo d'origine était au format **AVI** et c'est également le cas de la nouvelle. Cependant, la taille de la vidéo, originellement de **4.93 GO** est devenue **140 MO**.
-



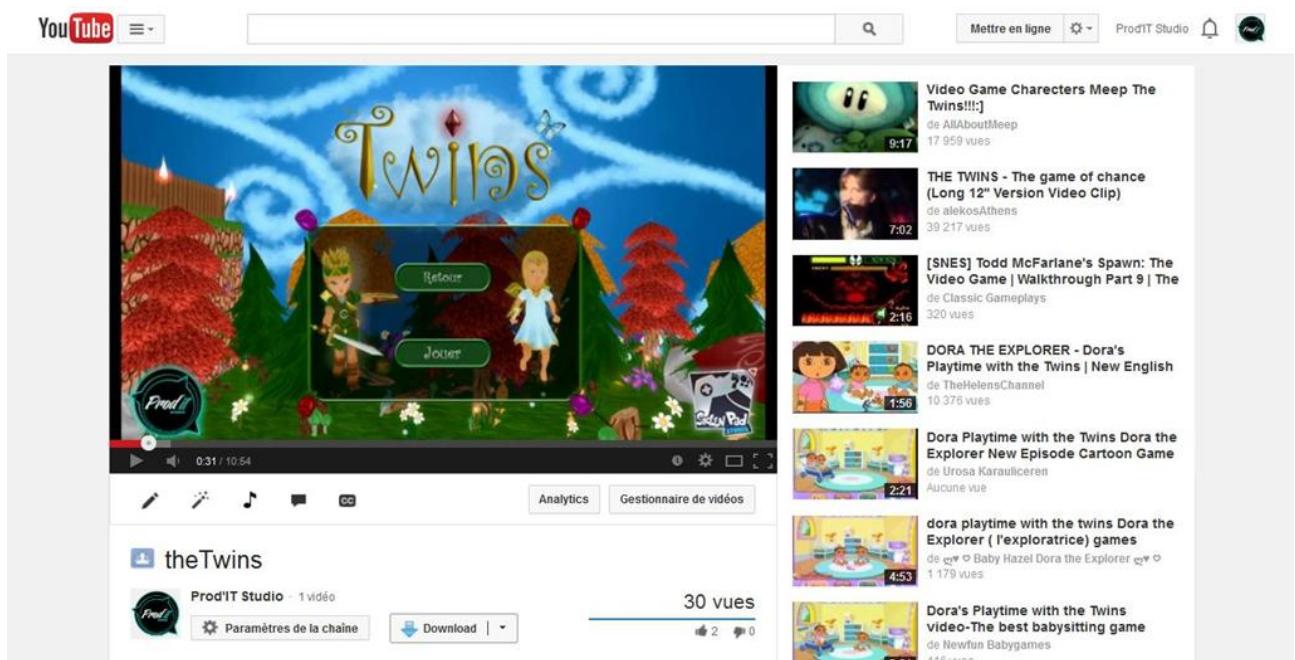
« Figure 172 : VSDC »

- La création de la chaîne sur Youtube pour l'entreprise Prod'IT Studio.



« Figure 173 : chaîne Youtube - Prod'IT »

- La mise en ligne de la vidéo compressée sur le compte de Prod'IT Studio.



« Figure 174 : vidéo de bande annonce "The Twins" sur Youtube»

6.3. La page Facebook

Avec le site web, les images du jeu et la vidéo sur « Youtube », nous avions suffisamment de contenu pour créer une page « Facebook ».



« Figure 175 : page Prod'IT sur facebook »

La page que nous avons créée présente le site web du jeu et son utilité, le jeu en lui-même, la chaîne Youtube dont elle partage les vidéos et surtout elle présente ce qu'est Prod'IT Studio.

Ash' Bouhadida a commenté ça..

Prod'IT Studio 2 mai, 22:02 ·

Prod'IT Studio possède désormais une chaîne YOUTUBE ainsi qu'un compte GOOGLE+ et une adresse Gmail :
-> proditstudio@gmail.com

Découvrez sur notre chaîne YOUTUBE la première bande annonce de "The Twins"... Afficher la suite

theTwins

The Twins est le tout premier jeu développé par l'équipe de Prod'IT Studio . The Twins est un jeu de plateforme et de réflexion qui se joue uniquement à deux.

[YOUTUBE.COM](#)

J'aime · Commenter · Partager · 1

« Figure 176 : une publication sur la page Prod'IT »

V. SPRINT 5 : Finalisation du projet



Enfin, après plus de trois mois de travail, l'aventure approchait de la fin. Cependant, avant de commencer à préparer la documentation, une toute dernière étape, un dernier sprint devait être réalisé : la **compilation** finale du projet. En effet pour que le jeu puisse être utilisé, il ne faut pas héberger le projet complet sur la page « **Facebook** » ou le site web, il faut uniquement héberger une version exécutable du produit. Un jeu vidéo pour ordinateur créé avec « **Unity** » peut être exporté au format « **.exe** » pour le système « **Windows** », sur d'autre format pour les systèmes « **Apple** » ou « **Unix** » ou encore sur un format global pour **navigateur web**. Ces formats permettant pour l'utilisateur d'exécuter le jeu sans avoir à utiliser le moteur physique. De plus, l'exécutable permet une encapsulation des composants du projet. Ainsi, l'utilisateur final du produit n'a pas accès aux objets 3D, aux textures ou aux codes sources. Cela constitue un besoin essentiel de protection du développeur.

1. Finalisation et ajout d'effets visuels et sonores

1.1. Une musique d'ambiance

La partie importante de l'ambiance créée dans un film ou un jeu vidéo est due à la musique et aux sons. L'ouïe combinée à la vue renforce la sensation de réalité du jeu.

Généralement, on trouve dans un jeu deux types de bandes sonores : la « **musique d'ambiance** » qui s'écoute indépendamment des actions réalisées dans le jeu et les « **effets sonores** » qui sont déclenchés par les actions des joueurs et de l'environnement. Par exemple lorsque le personnage marche, cela peut déclencher un effet sonore de bruits de pas qui s'arrêtera lorsque le joueur arrêtera d'appuyer sur le clavier.

Nous n'avons pas reçu de cours de musique. Cependant, nous avons tout de même des connaissances personnelles dans le domaine dont nous avons voulu faire bénéficier le projet. Ainsi nous n'avons pas créé tous les effets sonores mais **avons composé** une **musique d'ambiance** qui serait représentative de l'univers du jeu.

Ce travail reste néanmoins lié au projet et nous le détaillons ici car il concerne également la manipulation du logiciel « **Unity** » étudié en classe.

1.1.1. Le travail sur « **FL Studio** »

Pour créer cette musique d'ambiance, nous avons utilisé le logiciel « **FL Studio version 9** ».



« Figure 177 : FL Studio 9 »

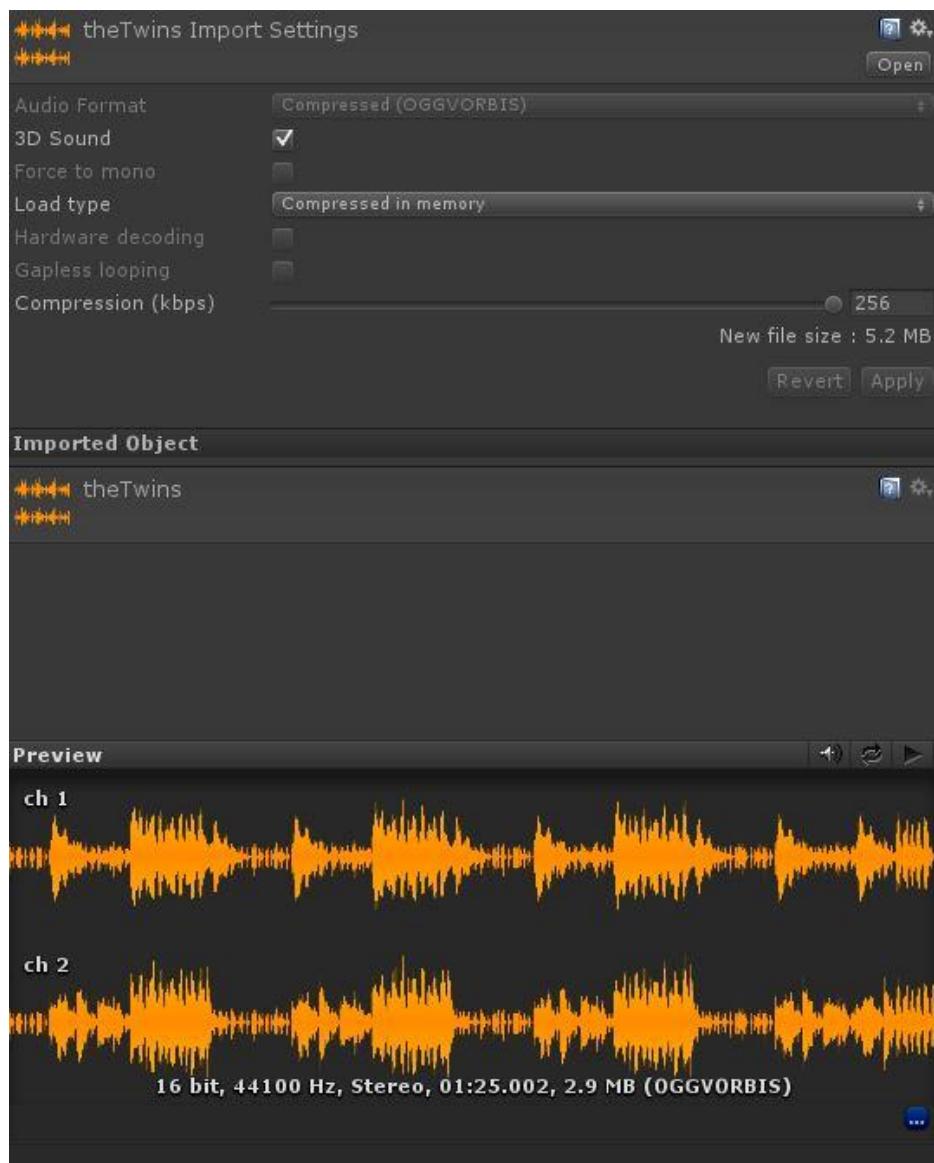
Ce logiciel a l'avantage de posséder une grande banque de sons. On y trouve par exemple le son de toutes les touches d'un piano ou de tous les coups d'une batterie. Cette banque de sons est visible sous forme d'une longue liste verticale à gauche de l'interface.

- La première étape de la composition d'une musique sur « FL Studio » consiste à choisir une vitesse (tempo) et un rythme dans les menus en haut de l'interface.
- Ensuite on doit choisir quels sont les sons que nous allons utiliser. Ces sons peuvent provenir de la banque ou être ajoutés à partir de l'ordinateur. Les sons choisis sont glissés dans la fenêtre tout à droite de l'écran : le créateur de « **Patterns** ».
- Les « **Patterns** » sont des petits assemblages de sons de longueur « **une mesure** ». Dans la fenêtre de création, chaque ligne représente un son choisi (avec des réglages de **volume** et de **stéréo**) et chaque colonne représente un « **quart de temps** ». On peut donc composé dans cette fenêtre des petits **rythmes** ou des petites **mélodies de quatre temps** qui constituent **une mesure**.
- Tous les **patterns** sont ensuite assemblés et réutilisés plusieurs fois pour créer la musique finale. Cet assemblage se fait dans la fenêtre au centre de l'écran. Pour que la musique puisse être utilisable dans le jeu, il faut qu'elle puisse boucler. En effet, on ne peut déterminer la durée de la partie de jeu et donc la musique dans le jeu risque d'arriver à sa fin. À ce moment là, il faut que le retour au début de la musique ne soit pas entendu par le joueur.
- Une fois la musique prête, on peut l'exporter au format « **MP3** » et l'importer dans le moteur **Unity**.

1.1.2. Le travail dans Unity 3D

Nous avons donc ajouté la musique « **twins.mp3** » dans les dossiers de la couche présentation du projet. Dans **Unity**, deux notions sont à prendre en compte lorsque l'on utilise des fichiers sonores :

- Les fichiers sonores sont classés en deux catégories : les « **sons 3D** » dont le volume dépend de la distance et les « **sons 2D** » qui n'en dépendent pas. Par exemple une musique d'ambiance qui doit être écouté avec le même volume où que l'on se place dans la scène doit être « **son 2D** » alors qu'au contraire, le son du crépitement d'un feu doit être de plus en plus fort lorsque le personnage s'en rapproche et sera donc un « **son 3D** ». Ce réglage se fait lors de l'importation du fichier :

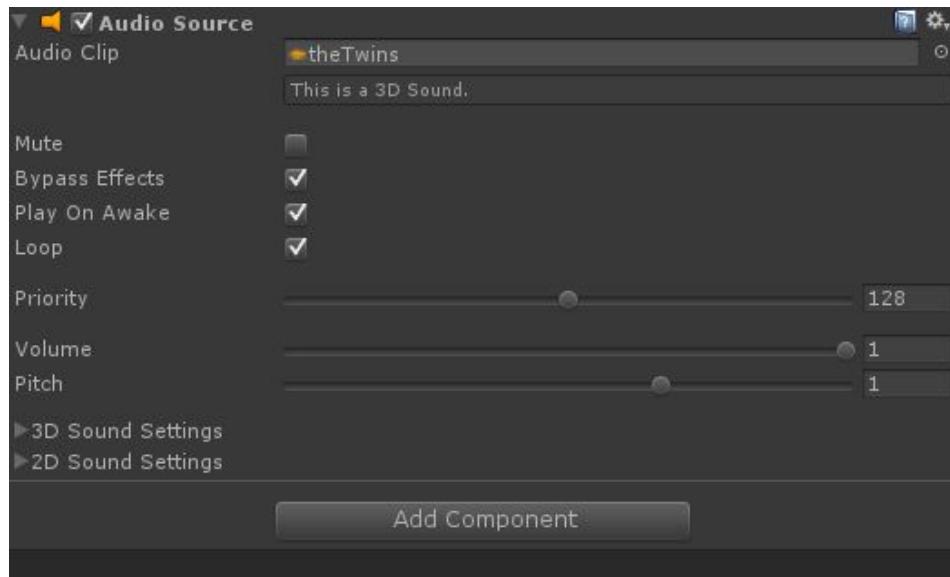


« Figure 178 : importation sur Unity »

- Pour qu'un son soit entendu sur la scène, il faut que la caméra possède un « component » d'écoute des sons : une instance de la classe « **AudioListener** ». Il faut également que l'objet 3D qui émet le son possède un « component » de production du son. Dans le cas d'une musique d'ambiance, aucun objet 3D n'est à l'origine du son et le « component » «

« **AudioSource** » est donc placé sur la caméra. Il faut également régler les variables « **loop** » et « **playOnAwake** » à « **true** » pour que la musique soit lue dès le début de la partie et qu'elle soit lue en continu.

-



« Figure 179 : composant " AudioSource " »

1.2. Un curseur de jeu

La plupart des jeux n'utilisent pas dans la partie le curseur classique de « **Windows** » qui par sa couleur et sa forme crée une coupure avec l'univers du jeu. **Unity** possède une fenêtre permettant de spécifier un curseur à utiliser. Cependant cette fonctionnalité n'est pas parfaite, il faut donc ajouter le curseur à l'aide de la programmation. On spécifie que le jeu doit utiliser ce « **curseur** » à l'aide de la fonction « **SetCursor()** » de la classe « **Cursor** ».

Le curseur a été dessiné de manière à ce qu'il corresponde à l'univers forestier et magique des Twins.

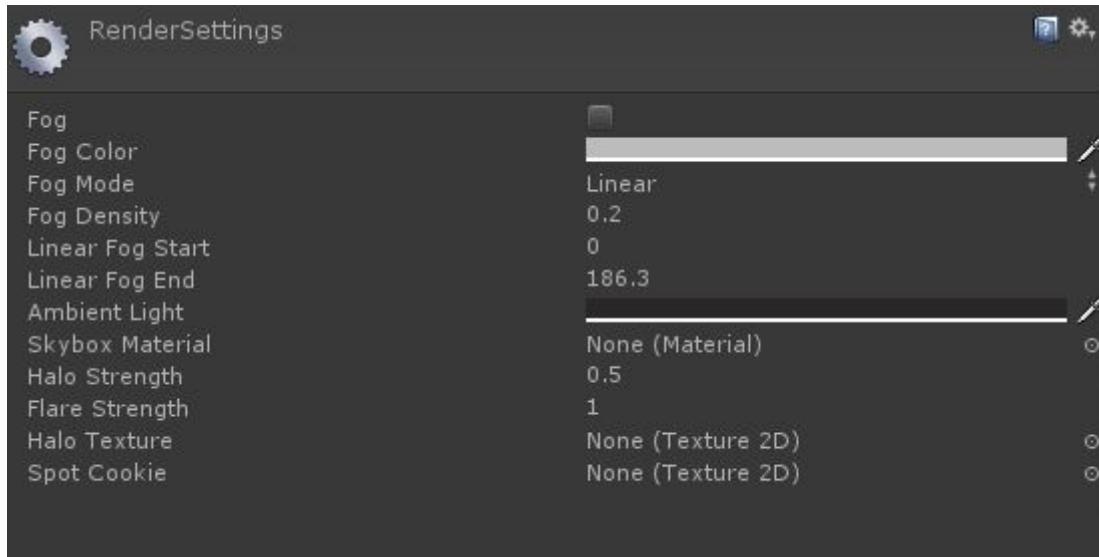


« Figure 180 : curseur du jeu »

2. Exportation de l'exécutable

Pour créer l'exécutable il faut réaliser une suite de réglages :

- Les réglages de la fenêtre « **RenderSetting** » sont utiles pour ajuster la beauté de l'image. On y trouve des paramètres de **luminosité** ou encore d'effets de **brouillard**.



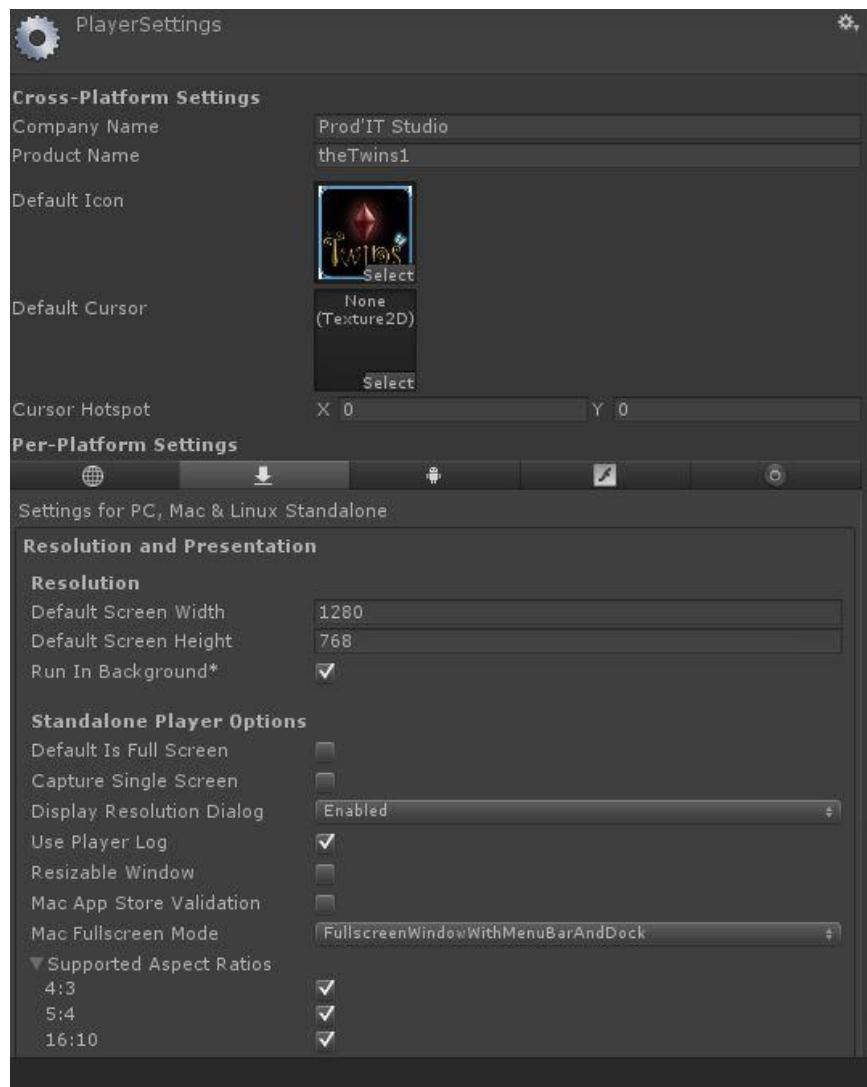
« Figure 181 : RenderSetting »

- Les réglages de la fenêtre « **QualitySetting** » servent à définir la qualité des **ombres**, des **textures**, la qualité de l'**« anti-aliasing »** (le nombre de pixel des contours des objets), ou l'utilisation du « **V-sync** » (fonctionnalité qui permet de camoufler le **balayage écran** lors du calcul d'une nouvelle image).



« Figure 182 : QualitySettings »

- Les réglages de la fenêtre « **PlayerSetting** » servent à définir l'icône du jeu, les formats d'écran supportés, ou encore la version de « **DirectX** » utilisée.



« Figure 183 : PlayerSettings »

Une fois tous les réglages terminés, nous avons pu exporter le projet en utilisant la fenêtre « **BuildSetting** ». Nous avons choisi d'exporter pour « **Windows** ». Après quelques minutes de compilation, l'icône du jeu apparut sur le bureau.



« Figure 184 : compilation »

Le produit était prêt, le projet fini et l'aventure de « The Twins » était donc achevée.

Conclusion

1. Bilan du projet

Objectifs du projet

Le premier objectif du projet en tant que produit, était la création d'un jeu vidéo multi-joueurs ainsi que celle d'un environnement de distribution du jeu tout en respectant un délai. Le jeu devait également avoir les caractéristiques d'être de type 2.5D dans un univers fantastique pour enfant et d'avoir en plus un intérêt éducatif. Ensuite, le deuxième objectif consistait à réaliser le premier objectif en respectant un processus de travail professionnel. Ainsi, nous avons développé le jeu « **The Twins** » dans cette optique.

Les délais ont été respectés et l'application est fonctionnelle, ces objectifs ont donc été atteints. En outre, l'utilisation des outils **Scrum** nous ont permis de valider l'attente d'un processus de travail professionnel.

Cependant le projet étant avant tout un projet de fin d'études universitaires, les objectifs pédagogiques prennent sur ceux liés au produit. Ainsi le principal objectif du projet était de développer une application dont la réalisation ferait intervenir le maximum de notions acquises dans les différentes matières étudiées. La réalisation devait nous forcer aussi à compléter ces connaissances en les approfondissant et effectivement cet objectif pédagogique a été également réalisé.

Difficultés et limites

Comme dans tout projet, nous avons rencontré à plusieurs reprises nos limites. Tout d'abord, en comparant notre produit fini avec les chefs-d'œuvre les plus réussis du monde du jeu vidéo, on peut facilement se rendre compte d'un large écart de niveau. Cela est du à trois principales raisons, nos compétences actuelles, le temps et le nombre de membres de notre équipe. En effet, un jeu vidéo professionnel est réalisé dans une période souvent supérieure à une année par une équipe surpassant généralement la centaine d'employés. Nous avons tenté de remplir à nous deux tous les métiers du jeu et certaines parties ont donc subi le manque de connaissances que nous avions pour ces spécialités. C'est le cas par exemple de la partie sonore du jeu.

Bénéfices du projet

The Twins constitue, en plus de son intérêt scolaire, le premier jeu vidéo réalisé par notre groupe **Prod'IT Studio**. Il est également le deuxième projet que nous ayons réalisé après le réseau professionnel « **proditnetwork.com** ».

2. Bilan personnel

Expérience acquise

La réalisation du projet a été pour nous une véritable et inoubliable expérience. Cela nous a permis de rencontrer et de travailler avec de véritables professionnels du domaine. Nous avons pu ainsi profiter de leurs connaissances. Nous avons aussi découvert le fonctionnement du travail en entreprise et les mécanismes du domaine du jeu notamment en Tunisie. Cela constitue notre première expérience professionnelle de longue durée.

Connaissances

Avant de commencer le projet, nos connaissances techniques étaient insuffisantes. Nous avons donc étudié un grand nombre de cours tel que des tutoriels vidéo (principalement ceux de [digitaltutors.com](#) et [formation-facile.fr](#)) ou encore des livres spécialisés (notamment ceux de [William Goldstone](#)). C'est en suivant tous ces cours que nous avons pu approfondir le développement 3D et la partie graphique du domaine.

Ensuite, les difficultés rencontrées durant le développement nous ont également conduits à chercher et apprendre des méthodes et techniques utilisées par les professionnels.

Enfin, le travail lors de l'encadrement nous a permis d'approfondir nos connaissances dans les domaines de la rédaction de rapports et d'articles ainsi que dans le domaine de la gestion de projet.

Perspectives avenir

Cette expérience a contribué à renforcer notre désir de travailler dans le domaine du jeu vidéo. Elle a également contribué à vouloir continuer de travailler ensemble et ainsi commencer **Prod'IT Studio**. Nous souhaitons d'ailleurs continuer le projet de The Twins cet été en commençant une version pour téléphone mobile.

Table des figures

« Figure 1 : les métiers du jeu vidéo »	10	
« Figure 2 : Child of Light »	14	
« Figure 3 : Limbo »	14	
« Figure 4 : Quake 3 »	15	
« Figure 5 : inFAMOUS : Second Son ».....	15	
« Figure 6 : Rochard ».....	16	
« Figure 7 : Guild Wars 2 »	18	
« Figure 8 : KillZone : Shadow Fall »	19	
« Figure 9 : Grand Thief Auto : San Andreas ».....	19	
« Figure 10 : une famille jouant à un jeu multi-joueurs ».....	21	
« Figure 11 : Mario Brothers Wii »	22	
« Figure 12 : un type d'ennemi du monde Mario ».....	23	
« Figure 13 : Trine »	24	
« Figure 14 : Giana Sisters »	24	
« Figure 15 : Link de la série Zelda »	« Figure 16 : Princesse Zelda de la série Zelda »	25
« Figure 17 : vidéo making of modular design - Trine 2 »	26	
« Figure 18 : modélisations low poly »	27	
« Figure 19 : modélisation high poly »	28	
« Figure 20 : Découpage des UV d'un cube »	29	
« Figure 21 : UV et textures d'un personnage 3D »	29	
« Figure 22 : Normal map d'un objet 3D »	30	
« Figure 23 : barres de santé »	31	
« Figure 24 : menu principale de Tom Oconnor ».....	32	
« Figure 25 : menu principal de Civilization 5 »	32	
« Figure 26 : objet mis en valeur »	33	
« Figure 27 : emplacements des GUI »	33	
« Figure 28 : Castle of Illusion ».....	34	
« Figure 29 : un post sur la page communautaire de Watch_Dogs ».....	36	
« Figure 30 : page d'accueil du site web de Frozenbyte ».....	37	
« Figure 31 : aperçu du site web de Trine 2 »	38	
« Figure 32 : scénario ».....	43	
« Figure 33 : schéma de navigation de The Twins »	44	
« Figure 34 : diagramme de cas d'utilisation ».....	45	
« Figure 35 : description textuelle du processus de connexion ».....	46	
« Figure 36 : description textuelle du jeu avec le personnage « Almas » »	47	
« Figure 37 : description textuelle du jeu avec le personnage « Umi » »	49	
« Figure 38 : diagramme de cas d'utilisation ».....	49	
« Figure 39 : modèle entité-association The Twins ».....	50	
« Figure 40 : diagramme de classe The Twins »	50	
« Figure 41 : Méta-modèle d'architecture conceptuelle».....	51	
« Figure 42 : diagramme d'activités de The Twins ».....	52	
« Figure 43 : diagramme d'activités pour « jouer avec le Frère » ».....	53	

« Figure 44 : diagramme d'activités pour « jouer avec la Sœur » »	54
« Figure 45 : diagramme d'activités concernant les actions des plantes mobiles »	55
« Figure 46 : diagramme d'activités système de score »	56
« Figure 47 : diagrammes de séquence »	57
« Figure 48 : Les couches de nos applications »	64
« Figure 49 : les dossiers du projet »	65
« Figure 50 : contenu du dossier COUCHE_PRESENTATION »	65
« Figure 51 : contenu du dossier COUCHE_DONNEES »	65
« Figure 52 : contenu du dossier COUCHE_METIER »	66
« Figure 53 : Les éléments clés de scrum »	66
« Figure 54 : schéma des éléments scrum »	67
« Figure 55 : Trello »	68
« Figure 56 : Almas »	77
« Figure 57 : Umi »	77
« Figure 58 : plante volante »	78
« Figure 59 : plante terrestre »	78
« Figure 60 : blueprint Umi »	79
« Figure 61 : blueprint Almas »	79
« Figure 62 : blueprint plante volante »	80
« Figure 63 : blueprint plante terrestre »	80
« Figure 64 : modélisation low poly – Almas »	82
« Figure 65 : limite de polygones – Almas »	82
« Figure 66 : modélisation low poly – Umi »	83
« Figure 67 : limite de polygones – Umi »	83
« Figure 68 : modélisation low poly - plante volante »	84
« Figure 69 : modélisation low poly - plante terrestre »	84
« Figure 70 : choix de couleurs - plante terrestre »	85
« Figure 71 : UV peints - plante terrestre »	86
« Figure 72 : UV peints - plante volante »	86
« Figure 73 : modélisation - plante terrestre »	87
« Figure 74 : modélisation - plante volante »	87
« Figure 75 : Un elfe de « Le Seigneur des Anneaux » »	88
« Figure 76 : choix de couleurs - Almas »	88
« Figure 77 : UV peints – Almas »	89
« Figure 78 : modélisation finale – Almas »	89
« Figure 79 : choix de couleurs – Umi »	90
« Figure 80 : UV peints – Umi »	91
« Figure 81 : modélisation finale – Umi »	91
« Figure 82 : rigging des personnages »	92
« Figure 83 : effets 2D »	93
« Figure 84 : importation des .fbx »	94
« Figure 85 : modification de shader »	95
« Figure 86 : type du rig »	95
« Figure 87 : affectation de bones »	96
« Figure 88 : fichier d'animations »	97

« Figure 89 : arbre Mecanim »	98
« Figure 90 : une liste de prefabs »	99
« Figure 91 : structure de scripts »	101
« Figure 92 : déclaration de variables optimisée »	101
« Figure 93 : déplacements »	102
« Figure 94 : saut »	103
« Figure 95 : trainée de coup d'épée »	103
« Figure 96 : sensation de frappe »	104
« Figure 97 : exemple de spritesheet »	104
« Figure 98 : système Shuriken »	105
« Figure 99 : pouvoir – téléportation »	106
« Figure 100 : dessin de cube »	107
« Figure 101 : cube créé »	108
« Figure 102 : déplacement du cube »	109
« Figure 103 : projectile de glace »	110
« Figure 104 : spritesheet effet de glace »	110
« Figure 105 : apparition en bulles »	111
« Figure 106 : activer / désactiver un héritage »	113
« Figure 107 : sendrate »	114
« Figure 108 : NetworkView »	114
« Figure 109 : requêtes SQL - création d'une base de données »	115
« Figure 110 : script contre les tentatives d'injection de code »	116
« Figure 111 : texture non tiled »	120
« Figure 112 : conception des éléments du décor »	121
« Figure 113 : modélisation sans blueprint 1 »	122
« Figure 114 : modélisation sans blueprint 2 »	122
« Figure 115 : texture regroupant plusieurs objets 3D »	123
« Figure 116 : résultat final après intégration de textures »	123
« Figure 117 : objets modulables du sol »	124
« Figure 118 : texture tiled »	124
« Figure 119 : conception de ciel »	124
« Figure 120 : ciel en volute »	125
« Figure 121 : composition de l'environnement »	125
« Figure 122 : halos »	126
« Figure 123 : optimisation au niveau des colliders »	127
« Figure 124 : layers de collisions »	128
« Figure 125 : pont qui s'effondre »	129
« Figure 126 : balançoire »	130
« Figure 127 : éboulement de rocher »	131
« Figure 128 : lac piégé »	132
« Figure 129 : lac gelé »	133
« Figure 130 : logo The Twins »	136
« Figure 131 : logo Prod'it »	137
« Figure 132 : Prod'it Studio »	137
« Figure 133 : structure spatiale de la fenêtre du jeu »	138

« Figure 134 : structure spatiale des menus »	139
« Figure 135 : GUI pour le compteur de joyaux »	140
« Figure 136 : GUI pour le compteur de temps ».....	140
« Figure 137 : barre de santé - "Umi" »	140
« Figure 138 : barre de santé "Almas" ».....	140
« Figure 139 : une des fenêtre du jeu »	141
« Figure 140 : bouton "Quitter" en surbrillance »	141
« Figure 141 : bouton "Quitter" »	141
« Figure 142 : bouton "Connexion" en surbrillance »	141
« Figure 143 : bouton "Connexion" ».....	141
« Figure 144 : interface - scène du jeu »	142
« Figure 145 : interface - sélection de personnages »	142
« Figure 146 : interface - écran de connexion ».....	143
« Figure 147 : importation d'image pour les GUI »	144
« Figure 148 : réglage GUI »	145
« Figure 149 : réglage GUIText »	145
« Figure 150 : étapes de barre de vie »	147
« Figure 151 : schéma de navigation étoilé »	150
« Figure 152 : ciel du jeu »	151
« Figure 153 : puce 1 »	151
« Figure 154 : puce 2 »	151
« Figure 155 : armes »	151
« Figure 156 : bulles ».....	151
« Figure 157 : silhouettes ».....	152
« Figure 158 : structure spatiale du site »	152
« Figure 159 : page index du site»	153
« Figure 160 : page "Galerie d'images" du site »	153
« Figure 161 : page "Bande annonce" du site »	154
« Figure 162 : page "Téléchargement" du site ».....	154
« Figure 163 : page "Communauté" du site ».....	155
« Figure 164 : page "Prod'IT Studio" du site »	155
« Figure 165 : structure des documents HTML »	157
« Figure 166 : fonction contre l'injection »	158
« Figure 167 : page d'erreur »	159
« Figure 168 : Jules Cesar »	160
« Figure 169 : cPanel – PlanetHoster ».....	160
« Figure 170 : FileZilla »	161
« Figure 171 : Fraps »	162
« Figure 172 : VSDC »	162
« Figure 173 : chaîne Youtube - Prod'IT »	163
« Figure 174 : vidéo de bande annonce "The Twins" sur Youtube».....	163
« Figure 175 : page Prod'IT sur facebook »	164
« Figure 176 : une publication sur la page Prod'IT »	164
« Figure 177 : FL Studio 9 ».....	166
« Figure 178 : importation sur Unity »	167

« Figure 179 : composant " AudioSource " »	168
« Figure 180 : curseur du jeu »	168
« Figure 181 : RenderSetting »	169
« Figure 182 : QualitySettings »	170
« Figure 183 : PlayerSettings »	171
« Figure 184 : compilation »	172

Table des tableaux

« Tableau 1 : backlog du produit »	71
« Tableau 2 : première partie du mois – sprint 1 ».....	74
« Tableau 3 : deuxième partie du mois – sprint 1 ».....	76
« Tableau 4: première partie du mois - sprint 2 ».....	118
« Tableau 5 : deuxième partie du mois - sprint 2 ».....	119
« Tableau 6 : première partie du mois - sprint 3 ».....	135
« Tableau 7 : deuxième partie du mois - sprint 4 ».....	149

Ludographie

(ArenaNet, 2012) Guild Wars 2. 2012. ArenaNet. PC, Mac. <https://www.guildwars2.com/fr>, consulté le 30 février 2014.

(Black Forest Games, 2012) Giana Sisters. 2012. Black Forest Games. PC, Mac, PS3, Xbox 360, Wii U. <http://gianasisterstwisteddreams.com>, consulté le 30 février 2014.

(Blizzard, 2005) World of Warcraft. 2005. Blizzard. PC, Mac. <http://eu.battle.net>, consulté le 30 février 2014.

(Capcom, 2014) Street Fighter IV. 2014. Capcom. PC, PS3, Xbox 360. <http://www.streetfighter.com>, consulté le 30 février 2014.

(Electronic Arts, 2006) Le Seigneur des Anneaux : La Bataille pour la Terre du Milieu II. 2006. Electronic Arts. PC, Xbox 360. <http://jeuxvideo.ea.com/bfme2>, consulté le 30 février 2014.

(Ensemble Studios, 2005) Age of Empire III. 2005. Ensemble Studios. PC, Mac. <http://www.ageofempires3.com>, consulté le 30 février 2014.

(Firaxis, 2010) Civilization V. 2010. Firaxis. PC, Mac. <http://www.civilization5.com>, consulté le 30 février 2014.

(Frozenbyte, 2011) Trine 2. 2011. Frozenbyte. PC, PS4, PS3. <http://frozenbyte.com/games/trine>, consulté le 30 février 2014.

(Guerrilla, 2013) Killzone : Shadow Fall. 2013. Guerrilla. PS4. <http://killzone3.killzone.com/kz3>, consulté le 30 février 2014.

(Hexa Drive, 2013) The Legend of Zelda : The Wind Waker HD. 2013. Hexa Drive. Wii U. <http://www.zelda.com/windwaker>, consulté le 30 février 2014.

(id Software, 1999) Quake III. 1999. id Software. PC.

(Konami, 2013) Pro Evolution Soccer 2014. 2013. Konami. PS3, Xbox 360, PSP.

(Microsoft, 2006) Flight Simulator X. 2006. Microsoft. PC.

(NetherRealm Studios, 2011) Mortal Kombat. 2011. NetherRealm Studios. PS3, Xbox 360, PSP Vita. <http://www.mortal-kombat.fr>, consulté le 30 février 2014.

(Neversoft, 2009) Guitar Hero : World Tour. 2009. Neversoft. PC, PS3, Xbox 360, Wii, PS2. <http://worldtour.guitarhero.com/fr>, consulté le 30 février 2014.

(Nintendo, 2013) Super Mario 3D World. 2013. Nintendo. Wii U, consulté le 30 février 2014.

(Playdead, 2011) Limbo. 2011. Playdead. PC, Mac, PS3, Xbox 360, iOS. <http://limbogame.org>, consulté le 30 février 2014.

(Recoil Games, 2011) Rochard. 2011. Recoil Games. PC. <http://www.rochardthegame.com>, consulté le 30 février 2014.

(Rockstar Games, 2004) Grand Theft Auto San Andreas. 2013. Rockstar Games. PS3, Xbox 360. <http://www.rockstargames.com/V>, consulté le 30 février 2014.

(Sega, 2013) Castle of Illusion starring Mickey Mouse. 2013. Sega. PS3, Xbox 360, Wii U, PSP Vita, iOS, consulté le 30 février 2014.

(Square Enix, 2013) Tomb Raider. 2013. SquareEnix. PC, PS4, PS3, Xbox One, Xbox 360.

<http://www.tomraider.com>, consulté le 30 février 2014.

(Square Enix, 2013) Final Fantasy XIV : A Realm Reborn. 2013. Square Enix. PC, PS4 et PS3.

<http://eu.finalfantasyxiv.com>, consulté le 30 février 2014.

(Sony, 2013) Resogun. 2013. Sony. PS4. <http://uk.playstation.com/resogun>, consulté le 30 février 2014.

(SuckerPunch, 2014) inFAMOUS Second Son. 2014. SuckerPunch. PS4.

<http://suckerpunch.playstation.com>, consulté le 30 février 2014.

(SuperBot Entertainment, 2012) Playsation All-Stars Battle Royale. 2012. SuperBot Entertainment.

PS3, PSP Vita. <http://www.playstationallstarsbattleroyale.com>, consulté le 30 février 2014.

(The Fullbright Company, 2013) Gone Home. 2013. The Fullbright Company. PC, Mac et Linux.

<http://www.gonehomegame.com>, consulté le 30 février 2014.

(Ubisoft Montreal, 2014) Watch Dogs. 2014. Ubisoft Montreal. PC, PS4, PS3, Xbox One, Xbox 360.

<http://watchdogs.ubi.com/watchdogs>, consulté le 30 février 2014.

(Ubisoft, 2014) Child of Light. 2014. Ubisoft. PC, PS3, PS4, Xbox One, Xbox 360.

<http://childoflight.ubi.com>, consulté le 30 février 2014.

Bibliographie

(Le Guide Complet de Scrum: Les Règles du Jeu, Juillet 2011) Ken Schwaber, Jeff Sutherland.

(Apprendre à développer des jeux 3D avec Unity, 9 juillet 2010) Will GoldStones, Pearson, Le programmeur.

(UML 2 par la pratique, 2009) Pascal Roques, Eyrolles.

(Apprendre le réseau avec Unity) www.formation-facile.fr.

(Modeling Low Polygon Game Characters in 3ds Max) www.digitaltutors.com.

(A multi-layer framework for 3D Videogames Evaluation), Achref Bouhadida, Anas Neumann, Jihen MALEK, to be submitted in “Entertainment Computing Journal”.