Below

MTV

Rapport de Projet

Mai 2021

Ernest Bardon Edgar Delaporte

Matthieu Colin Jules Diaz

Table des matières

0.1 Répartition générale des tâches

E = En charge / S = Suppléant

	Ernest	Matthieu	Edgar	Jules
Graphisme				
Map		E	S	
Effets	E	S		
Personnage	E			S
Objets	E		S	
Son				
Ambiance sonore	S			E
Menus				
Accueil	S	S		E
Options		S		E
Réseau				
Photon		E		S
IA				
Monstres	S		E	S
Boss		S	E	
Mécaniques de jeu				
Implémentation maps (Spawns, objets ...)			S	E
Jouabilité (contrôles, collisions)	S		E	
Mode Solo			E	S
Média				
Site internet	S	E	S	
Vidéo Présentation		E		

1 Genèse

1.1 Création du groupe

MTW a débuté au cours du mois de décembre 2020, une idée de jeu de cartes était débattue entre **Jules** et **Edgar**.
L'idée plaisait, et puisque Jules projetait déjà de travailler avec **Matthieu**, ce dernier s'est joint naturellement au groupe.
Enfin, **Ernest** a été contacté quelques jours plus tard et s'est montré également intéressé par l'idée originelle du jeu.
Une fois complet, les membres du groupes se sont réunis le soir même pour discuter de ce qu'allait être le projet.
Il a été décidé que le groupe se nommerait **MTW**, en référence à une blague sur **OCamL** qui ne sera pas révélée ici.
Une fois prêts, il était temps de commencer à construire le jeu.

1.2 Recherche du jeu

Comme dit plus haut, le jeu était au départ pensé comme un jeu de cartes. Cette idée a été assez rapidement abandonnée car elle posait trop de contraintes aux vues des consignes de développement qui nous étaient imposées.
Beaucoup d'autres types de jeu ont alors été proposés, allant du **Action RPG** au jeu de stratégie.
Finalement, le **Rogue-Lite** a été retenu car il offrait de grandes libertés de **gamedesign** et apportait assez peu de contraintes formelles.
Nous avions en tête un jeu dans le style de **The Binding of Isaac** mais pensant que cela s'adapterait mal à la 3D et au multijoueurs, nous avons basculé sur un **gameplay** plus proche de **DOOM** (si ce n'est que notre jeu allait être à la troisième personne).
Il y a également eu beaucoup de débats sur ce que devrait être l'enrobage du jeu (thématiques graphiques, ambiance, histoire...).
Après avoir longuement hésité avec un univers inspiré d' **Alien**, nous nous sommes finalement arrêtés sur un univers de **Dark Fantasy** dans l'inspiration de **Darkest Dungeon**.
Enfin, il nous fallait un nom pour notre jeu.
"**Below**" a été très vite retenu, donnant un indice sur l'ambiance du jeu tout en laissant une part de mystère.

1.3 Contenu du jeu

Une fois que nous avions la ligne directrice pour **Below**, nous avons défini les points principaux sur lesquels il allait falloir se focaliser. Le multijoueurs est la fonctionnalité principale, qui allait nécessiter à la fois le plus de temps et d'attention. En effet, le jeu reposant sur des mécaniques de **versus**, le réseau se doit d'être d'une solidité à tout épreuve afin qu'il soit agréable.
Ensuite, le deuxième point fondamental est le **gameplay** en lui-même. Un jeu ne peut être bon si les fonctionnalités de base telles que les mouvements ou les attaques ne sont ni réfléchis ni adaptés à son style.
Ce point n'est pas le plus difficile techniquement mais c'est le plus crucial en terme de **gamedesign**. L'implémentation de l'**IA** est également importante car cela participe grandement à rendre le jeu dynamique et à créer de la difficulté. Il était donc nécessaire d'y apporter un soin tout particulier.
Enfin, nous tenions énormément à un système d'objets aléatoires variés, qui permettrait une diversification des parties et donc une grande rejouabilité. Il fallait donc créer un grand nombre d'objets différents afin de rendre cela possible.

Une fois toutes ces idées en tête, nous étions à même de mettre en place un cahier des charges.

2 Retour sur le cahier des charges

2.1 Histoire et intérêt

2.1.1 Histoire

Lors de la création de notre groupe, le type de jeu du projet n'était pas encore fixé. Après plusieurs discussions, nous avons finalement décider de nous inspirer largement des jeux de type **Rogue Lite**. Pour rester conformes aux contraintes qui nous étaient données, nous avons pris la décision de tordre le concept de **Rogue Lite** pour que celui-ci soit multijoueurs.
Le genre du **Rogue-Lite** (ou **Rogue Like-Like**) est un type de jeu vidéo solo avec une difficulté très souvent supérieure à la moyenne. Le joueur est transporté dans un monde hostile généré de manière procédurale. Le principe du jeu est d'apprendre de ses erreurs en le recommençant autant de fois qu'il le faut pour réussir (c'est ce qu'on appelle la mécanique du **Die and Retry**).
Malgré ces caractéristiques, ce genre reste assez flou. Les amateurs s'accordent à dire que les origines de ce genre viennent du jeu **Diablo** sorti en 1997 et développé par **Blizzard**. L'histoire de ce genre est cependant fortement liée au **Rogue-Like**. Le **Rogue-Like** est un type de **RPG** (jeu de rôle) qui s'inspire principalement du jeu **Colossal Cave Adventure** mais aussi du jeu **Rogue**. Celui-ci est très proche du concept de **Rogue-Lite** avec notamment la mécanique du **Die and Retry** et la génération procédurale. Mais contrairement au **Rogue Lite**, ce type de jeu est centré sur les **RPG** tour par tour. Le **Rogue Lite** est lui plus centré sur les réflexes du joueur ainsi que sa réflexion.
Nous pouvons citer **The Binding of Isaac**, **FTL : Faster than Light** ou encore **Spelunky** qui sont trois grands noms du genre.

2.1.2 Intérêt

Le **Rogue-Lite** est un genre très vaste qui présente donc de nombreux intérêts. Tout d'abord la génération procédurale offre au joueur la possibilité de jouer de nombreuses parties, toutes différentes les unes des autres, sans perdre goût au jeu. De plus, la mécanique du **Die and Retry** force le joueur à se dépasser pour progresser. Le joueur doit apprendre tout seul de ses erreurs car le jeu n'offre pas de phase d'apprentissage.

Le joueur est poussé à expérimenter, réfléchir et tester des stratégies personnelles afin de progresser et surmonter les épreuves du jeu. L'aléatoire ajoute aussi une difficulté au jeu car le joueur ne peut pas se baser sur ses connaissances, le jeu étant différent à chaque partie.
Enfin, pour ce qui est de **Below**, nous avons choisi d'ajouter un mode multijoueurs. Cela nous permet, en plus de respecter les contraintes du projet, de varier toujours plus les mécaniques de jeu avec l'ajout d'un autre joueur. Ainsi, chaque joueur devra faire attention à l'environnement mais aussi aux actions de son adversaire.

2.2 Éléments fondamentaux

Lors de l'écriture du cahier des charges, aucun membre du groupe n'avait utilisé **Unity** auparavant et encore moins entrepris d'aussi grand projet. Cependant, étant tout à fait conscients de notre manque d'expérience nous avons préféré nous donner des objectifs atteignables mais non moins exigeants.

2.2.1 Réseaux

Pour ce qui est du réseau nous voulions un système où un des joueurs devrait créer une salle avec un nom au choix puis le second devant le rejoindre. Une fois cette étape terminée le maître de la salle lance la partie et le jeu commence.
Pour ce faire nous avons donc choisi d'utiliser l'**asset Photon Unity Engine** qui dans sa version gratuite peut nous permettre de créer jusqu'à 10 salles simultanément.

2.2.2 IA

Nous avions aussi imaginé dans le cahier des charges des **Intelligences Artificielles** qui serviraient d'ennemies afin de rendre les salles et le jeu plus vivants. En effet, si l'on devait attendre de trouver le joueur adverse pour commencer à combattre le jeu serait bien vide...
Nous avons ainsi réfléchi à plusieurs catégories d'ennemi "classique" comme un ennemi qui court vers le joueur le plus proche pour l'attaquer, un autre qui vole et qui tire sur le joueur, un petit très rapide mais faible etc...

Pour ce qui est du boss du mode "un joueur", une **Intelligence Artificielle** un peu plus développée était envisagée. Rendre le combat plus intéressant et plus ardu permettrait de faire ressentir une vraie satisfaction pour le joueur lors de sa victoire.

2.2.3 Mouvements et caméra

Les mouvements que nous voulions pour notre personnage étaient simples : avancer, reculer, aller à gauche, aller à droite, courir et sauter. Ces déplacements sont basiques mais efficaces, permettant ainsi de jouer simplement sans pour autant se sentir bridé.
Du côté des attaques nous avions imaginé un large panel d'armes au corps-à-corps et d'armes à distance. Chaque joueur aurait une arme de chaque type qu'il pourrait changer selon sa préférence parmi les autres armes trouvées dans les coffres durant la partie.

Finalement pour ce qui est de la caméra elle serait en vue troisième personne, assez rapprochée du personnage.
De plus, les déplacements dépendraient de celle-ci, c'est-à-dire que le protagoniste serait toujours orienté en direction de là où pointe la caméra.

2.2.4 Histoire et galerie

Finalement, nous pensions ajouter quelques éléments liés à l'histoire comme des pages de manuscrits cachées un peu partout dans le niveau du mode "un joueur". Celles-ci raconteraient l'histoire d'aventuriers venus avant nous et dont le sort serait inconnu.
Mais aussi une galerie d'objets accessible depuis le menu où nous pourrions observer les objets déjà trouvés. On pourrait y voir le dessin de chaque objet en plus grand ainsi que son utilité et son histoire s'il en a une.

2.3 Structure du jeu

Le déroulement de **Below** se divise en trois parties. Ces dernières vont de ce fait déteindre sur l'architecture de la carte qui sera divisée en trois. La carte est symétrique, chaque partie est composée

de salles différentes. Les joueurs commencent de part et d'autre.

La première partie de jeu pourrait s'apparenter à une phase de préparation. Durant cette première phase de jeu chaque joueur essaie de récupérer le maximum d'objets afin d'améliorer ses compétences. Ces objets sont disposés aléatoirement sur la carte et peuvent être récupérés en battant les monstres qui les protègent. Les joueurs sont chacun dans une zone différente de la carte et ne peuvent donc au départ interagir entre eux mais cela ne durera pas.

En effet, dans la deuxième partie de jeu les joueurs ont la possibilité de passer dans la zone de la carte qui était avant réservée à d'autres joueurs. Ce changement permet d'offrir aux joueurs de nouvelles possibilités. Ils ont le choix entre, rester dans leurs zones pour améliorer leur équipement ou passer dans la partie ennemie pour voler les objets qui s'y trouvent. S'ils prennent ce pari ils pourront même combattre les joueurs adverses pour les empêcher d'obtenir de nouveaux objets.

La dernière phase de jeu se déroule dans une partie pour l'instant inexplorée de la carte. Il s'agit de la salle aux trésors se trouvant au centre de la carte. Cette dernière regorge d'objets puissants mais aussi de dangereux ennemis. Au début de la troisième phase, les portes qui scellaient la salle aux trésors s'ouvrent permettant aux joueurs d'y accéder mais dans cette zone la mort est définitive. Les joueurs ont de nouveau un dilemme : aller dans la salle aux trésors pour y récupérer des objets uniques au risque de tomber sur de nombreux et puissants monstres ou alors rester dans sa zone et laisser la possibilité à ses adversaires d'aller dans la salle aux trésors.

Enfin si aucun des joueurs ne prend l'avantage, la partie rentre dans une ultime phase qui va départager les joueurs. Ils se retrouvent tous téléportés au centre de la salle aux trésors, dans une arène où ils combattront à mort !

2.4 Choix des logiciels

La réalisation de **Below** a nécessité l'utilisation de nombreux logiciels. Le premier, le plus important, est **Unity**, notre moteur de jeu. Afin

d'utiliser pleinement **Unity** nous avions besoin d'un environnement de développement supportant le **C Rider** édité par **jetBrain**.
Pour développer le site web nous avons utilisé les logiciels **Visual Studio Code** de **Microsoft** et son équivalent de **jetBrain**, **PhpStorm**, tous deux supportant l'**html**, le **css** et le **javaScript** .
Les monstres et le coffre présents dans **Below** seront entièrement faits par **MTW**. Nous avons fait des maquettes à l'aide d'**Illustrator** puis les avons transformées en maquettes 3D grâce à un logiciel de modélisation, **Blender**.

2.5 Choix des graphismes

2.5.1 Présentation des graphismes du jeu

Tout comme le choix du type de jeu, l'aspect graphique est passé par différentes idées avant qu'une ne soit retenue. Nous nous sommes demandés quels seraient les graphismes les plus adaptés à notre vision du jeu. Parmi les propositions de chacun, deux sortaient du lot : un aspect médiéval de donjon sombre ou un aspect futuriste dans un vaisseau spatial. Nous avons finalement choisi la première option car plus adaptée à notre vision du jeu. Le jeu se déroule ainsi dans un donjon lugubre riche en objets magiques et peuplé de créatures surnaturelles.
Un **gameplay** à l'intérieur d'un donjon rend l'ambiance pesante plus facile à mettre en place car le joueur n'a aucun repère extérieur, il se sent donc enfermé et oppressé dans le donjon.

Pour le style graphique nous avons décidé de faire un mélange de 2D et de 3D.

Tout d'abord nous avons choisi un style graphique **low poly** pour la majorité du jeu ce qui nous permet de créer des modèles 3D assez facilement, cela malgré notre manque de compétences dans la modélisation 3D lors du choix du style graphique. Ce choix est de plus très utile pour garder un jeu fluide grâce à des modèles 3D peu détaillés. Nous avons décidé d'utiliser **Blender** pour la modélisation, un logiciel connu, très utile pour apprendre les bases.
Enfin, la partie du jeu 2D, c'est-à-dire les **sprites** d'objets, la barre de vie ou encore le menu principal du jeu ont été dessinés sur tablette graphique tout en gardant la même direction artistique.

Nous avons choisi le logiciel **Aggie** puis **Illustrator** pour cette partie du style graphique.

2.5.2 Présentation des logos

Nous avions formé notre groupe en avance ce qui nous a permis de produire les deux premiers logos du groupe avant le cahier de charges.
Tout d'abord nous avons designé le logo du groupe. Celui-ci est une simple fusion des lettres du nom du groupe de manière symétrique et polygonale.
Dans un second temps, nous avons fait le logo du jeu **Below**. Sur celui-ci est représenté le nom du jeu écrit en lettres manuscrites médiévales avec une banderole les traversant. Cela présente l'ambiance du jeu, c'est-à-dire l'aspect graphique médiéval sombre. Les couleurs du logo sont de plus en harmonie avec l'ambiance de jeu.

3 Mise en place des bases

3.1 Implémentation des logiciels

Dans notre cahier des charges nous avons choisi l'utilisation de différents logiciels, cependant nous n'avions pas décidé de quelles versions nous aurions besoin.
Cette problématique s'est vite retrouvée très importante car nous nous sommes rendus compte de la nécessité d'avoir tous les mêmes versions des logiciels. De plus nos logiciels devaient être compatibles entre eux. Nous avions aussi prévu pour les graphismes d'utiliser des **assets** directement récupérés de l'**Asset Store** de **Unity**, or chacun d'entre eux n'est compatible qu'avec une certaine version du logiciel. C'est donc pour cela que nous avons choisi une version d'**Unity** dite **LTS (Long Term Support)** qui est compatible avec la plupart des **assets** présents et futurs.
En ce qui concerne les autres logiciels nous avons utilisé les versions les plus récentes.

3.2 Workflow

Dans ce contexte de pandémie mondiale, nous ne pouvions nous retrouver régulièrement pour travailler sur le jeu. Il nous a donc fallu mettre en place des outils afin de rendre le travail à distance le plus pratique possible.
La première étape fut la mise en place d'un serveur **Discord** sur lequel nous pouvions nous retrouver pour travailler et échanger des ressources.
Ensuite, nous avions besoin d'un espace pour stocker et partager notre travail. C'est pourquoi nous avons créé un repository **Git** auquel nous accédons grâce à **Git Kraken**.
Cela nous permet de travailler à plusieurs sur le même projet **Unity** et sur les mêmes scripts sans craindre les problèmes de **Merge** (en théorie tout du moins).
La possibilité de créer des branches afin de séparer les différentes parties du développement s'est également révélée très utile.
La prise en mains de **Git Kraken** n'a pas été simple pour tout le monde mais au bout du compte, chaque membre est parvenu à utiliser cet outil à sa manière.
Enfin, quand il nous a fallu produire un document papier, nous avons

utilisé **Overleaf**. Ce logiciel permet d'éditer un même document texte à quatre de manière très simple et rapide, en plus de gérer automatiquement des points comme l'indentation ou la création du sommaire. Nous avons ainsi gagné un temps précieux, bien que l'utilisation du **LateX** n'est pas été simple pour tout le monde.

3.3 Découpage du travail

Pour ce qui est de la répartition des tâches, nous voulions que chaque personne ait travaillé de près ou de loin sur chaque domaine. Cependant il a quand même fallu attribuer aux membres du groupe des sujets un peu plus précis.

Ernest, de part son talent pour le dessin, s'est vu attribuer la majeure partie des graphismes. Il s'est donc focalisé sur le design des personnages et monstres ainsi que celui des objets.
Pour le reste, personne n'ayant d'expérience particulière dans un domaine, nous avons réparti les tâches encore non attribuées au bon vouloir de chacun suivant les inspirations et les envies.

4 Première phase de développement

4.1 Depuis une scène vide...

La première fois que nous avons ouvert **Unity**, nous avons été confrontés à notre première scène vide.
Tout était à faire.
Ne sachant pas par où commencer, nous avons visionné des tutoriels sur **Youtube** et lu de nombreuses pages de la documentation de **Unity** afin de savoir quoi faire.
Une fois les première bases connues, nous avons placé un plan dans notre scène vide et avons commencé à implémenter la caméra.
En effet, sans caméra **Unity** n'affiche jamais à l'utilisateur ce qui se passe dans le jeu. Ce qui n'est pas très pratique ni pour jouer ni pour tester son code.
Après le placement de la caméra sur notre plan nous pouvions compiler le jeu et le lancer. Ce n'était pas très intéressant cela-dit car cela se résumait à un plan fixe sur du vide.
L'étape suivante a été de donner à l'utilisateur la possibilité d'orienter la caméra. Pour ce faire, nous avons ajouté au **GameObjet** de la caméra le module **FreeLook**.
C'est un module intégré à **Unity** qui permet au joueur de changer l'angle de la caméra à l'aide de sa souris.
Désormais nous pouvions déplacer une caméra, il ne manquait plus que quelque chose à regarder.

Nous avons donc dû créer une scène de "test" avec quelques éléments afin de pouvoir se repérer dans l'espace. Cela étant fait, il était temps de s'atteler aux mouvements du joueur.

4.2 Mouvements de base

Avec la mise en place des mouvements, nous avons commencé à éditer nos premiers scripts C# complexes.
Nous pensions au départ modifier simplement les coordonnées du joueur selon les touches pressées. Mais cela ne pouvait pas fonctionner correctement.
En effet, il nous a fallu utiliser une classe spécifique à **Unity** nommée **Vector3** afin d'appliquer des forces sur un objet de manière dynamique et cohérente avec le reste des éléments du jeu.

Ainsi, lorsque le joueur appuie sur une touche de mouvement (ZQSD), un nouveau **Vector3** s'ajoute au composant **transform** de l'objet représentant le joueur.
Cela résulte en une force appliquée sur le modèle, ce qui le fait bouger dans la direction voulue. Ces forces peuvent se superposer, ainsi si le joueur appuie sur Z et Q, il avancera en diagonale haut-gauche.
En ce qui concerne la chute, nous avons simplement utilisé l'implémentation par défaut de la gravité dans **Unity**. Ces scripts de mouvements ont ensuite été attachés à la caméra. Le joueur pouvait donc désormais se déplacer librement dans la scène.

Mais notre projet n'est pas un jeu à la première personne, nous avons donc du trouver un modèle afin d'avoir un personnage à suivre.
En cherchant dans l'**Asset Store** de **Unity**, nous avons trouvé un modèle de personnage basique (nommé "MaleDummy") qui allait nous servir pour nos tests avant d'avoir notre modèle unique. Pour ce faire, nous avons créé notre premier objet préfabriqué : le modèle du MaleDummy auquel est attaché la caméra.
Nous avons également verrouillé les rotations de la caméra relatives aux mouvements du MaleDummy afin que seule la souris soit capable de l'orienter.
Une fois cela fait, nous avions notre personnage à la troisième personne que nous pouvions déplacer dans notre scène.

4.3 Réseau

Le multijoueurs est au centre de **Below**, nous avons donc commencé son développement très tôt.
Comme nous l'avions énoncé dans le cahier des charges l'implémentation du réseau s'est faite grâce à **Photon**.
Photon permet de grandement simplifier la création du multijoueurs en hébergeant nos parties et donnant accès à une multitude d'outils pour gérer notre réseau.
La première étape dans le développement du réseau a donc été de créer un projet **Photon** et de le lier avec notre projet **Unity** où nous développons le jeu.
Cette association a été très simple : nous avons simplement dû ajouter l'*asset* **Photon** dans Unity et entrer un code fourni lors de la création du projet.

Une fois **Photon** ajouté à notre projet nous n'avions plus qu'à coder les comportements que nous voulions pour le multijoueurs. Pour cela nous avons dû ajouter à chaque élément dynamique du décor différents composants pour les rendre visibles des autres joueurs. Ces composants sont **PhotonView**, **TransformView** et **AnimatorView**, pour les objets animés.
Nous avons aussi dû implémenter un menu primitif pour créer et rejoindre les parties en ligne.

4.4 Jouabilité avancée

Les mouvements et le réseau désormais opérationnels, il nous a fallu nous intéresser aux fonctionnalités "avancées" du jeu .

4.4.1 Course

Afin que le jeu soit dynamique, il est indispensable que le personnage du joueur puisse courir. Pour cela, il suffit d'ajouter quelques lignes au script des mouvements.
Ainsi, si la touche **SHIFT** est pressée pendant un mouvement, les valeurs du **Vector3** lié à ce dernier augmentent progressivement.
Cela se traduit à l'écran par un mouvement plus rapide.
Il a fallu limiter la valeur de la vitesse, sans quoi le joueur accélérerait indéfiniment jusqu'à sortir des limites de la scène.

4.4.2 Sauts

Le saut fut plus complexe à implémenter que la course.
En effet, pour qu'un saut soit réaliste, sa vitesse doit décroître sur la montée jusqu'à atteindre zéro puis augmenter au fur et à mesure de la chute jusqu'à ce que le joueur touche le sol.
Le script de saut que nous avons réalisé marche comme suit :

- Lorsque le joueur appuie sur **ESPACE** et qu'il y a un sol sous ses pieds, la gravité est désactivée pour lui et un **Vector3** est appliqué sur son **transform** afin qu'il monte.
- La vitesse de montée diminue progressivement jusqu'à atteindre zéro.
- La gravité est réactivée et le joueur commence à redescendre.

- La vitesse de chute augmente jusqu'à ce que le joueur touche le sol.

Cela a demandé beaucoup de temps et de tests afin que le saut soit cohérent. Des variables comme la durée du saut, sa hauteur ou la vitesse des mouvements ont été complexes à établir. Nous nous sommes donc basés sur des références de jeux connus.
Nous avons voulu que le saut soit légèrement flottant afin de pouvoir effectuer quelques actions dans les airs.

4.4.3 Attaques rapprochées

A ce moment-là, nous avions toutes les options de déplacement souhaitées, il était donc temps de travailler sur le combat.
Pour cela nous avons dû nous intéresser à la classe **Collider** de **Unity**.
En effet, quand le joueur utilise le clic droit de la souris, cela crée un **SphereCollider** devant lui qui va récupérer les objets avec lesquels il est en contact. C'est la *hitbox* de l'attaque.
En l'occurrence, cela récupère les **Colliders** des objets juste en face du joueur et les projettent dans la direction opposée.
Évidement, le joueur ne peut pas attaquer en boucle, il y a un temps de pause d'au moins une seconde entre chaque coup porté.

4.4.4 Attaques à distance

Il nous a ensuite fallu incorporer les attaques à distance.
Nous avons pour cela eu besoin de la classe **RayCast** de **Unity** qui permet de tracer des "droites" invisibles dans les scènes afin de donner des trajectoires.
Ainsi, lorsque le joueur appuie sur le clic gauche, un **RayCast** est tracé entre l'axe de la caméra et le joueur. Ensuite, un objet préfabriqué "Shot" est instancié au niveau du joueur puis propulsé dans le sens du **RayCast**.
Lorsque cet objet "Shot" rencontre un autre objet, il disparait et laisse à la place un **Collider** avec les mêmes propriétés que celui de l'attaque rapprochée.
Afin de dynamiser le tout, nous avons associé aux tirs et au **Collider** des animations de boules de feu et d'explosions trouvées sur l'**Asset Store**. L'implémentation des tirs a demandé beaucoup de temps et de patience car il a fallu envisager de très nombreux paramètres

(orientation du modèle du joueur, orientation de la caméra, couches de collision, layers...).

Une fois les deux types d'attaque implémentés, nous avons ajouté des variables de points de vie aux joueurs afin qu'ils puissent s'attaquer.
A ce moment du développement, rien n'est prévu lorsque le joueur meurt, alors cela arrête l'exécution du programme.

4.5 Animations

Malgré toutes ces fonctionnalités ajoutées, le jeu était toujours très morne car le modèle du joueur n'était pas animé.
Nous avons donc mis en place un **Animator**.
Un **Animator** est un réseau d'animations reliées entre elles par des conditions (booléens).

Nous vous conseillons de vous référer au schéma de l'**Animator** présent dans l'annexe pour cette partie.

L'**Animator** commence au stade **Entry** et se dirige à l'état par défaut appelé "Locomotion" qui contient les animations des mouvements de base (la marche).
On se déplace ensuite entre les animations via les conditions déterminées. Par exemple le booléen représenté par la flèche allant de **Locomotion** à **Run** représente le fait que la touche **SHIFT** soit ou non pressée.
La case "AnyState" représente toutes les autres cases à la fois, sa flèche peut donc être atteinte à partir de n'importe quel endroit de l'**Animator**. La case "Exit" n'est jamais atteinte car les animations du joueur ne s'arrêtent jamais.
Lorsque l'**Animator** empreinte une flèche, il effectue une transition dans les animations. C'est-à-dire qu'il ne change pas brutalement les articulations du joueur mais les déplace progressivement afin que le mouvement soit fluide et esthétique.
Ce composant **Animator** est lié à l'objet préfabriqué du joueur afin que les animations soient correctement appliquées à son modèle.

4.6 Illustrations

Toutes nos illustrations ont été créées spécialement pour le jeu et par notre groupe.

4.6.1 Bestiaire

Nous avons, au cours de la première phase de développement, décidé de faire des illustrations 2D du bestiaire de **Below** pour deux raisons.
Tout d'abord, cela nous a aidé à la modélisation 3D car elle apporte une base solide de direction artistique.
Et d'autre part, ces illustrations se sont avérées utiles pour la présentation du jeu, sur le site notamment, car elles donnent une idée aux joueurs de l'ambiance du jeu, sans en révéler le contenu.
Ces illustrations sont faites sur **Illustrator** d'**Adobe** à l'aide d'une tablette graphique. Durant cette phase de développement, nous avons pu faire 80% des illustrations du Bestiaire prévues originellement, avec notamment le personnage jouable et quelques monstres.
Vous retrouverez ces illustrations dans l'annexe.

4.6.2 Autres

Nous avons également produit des illustrations destinées à être utilisées directement dans le jeu.
Pour commencer, nous avons ajouté très rapidement une illustration pour la barre de vie, dont nous reparlerons plus tard.
Puis, nous avons fait un croquis du menu afin de visualiser ce à quoi il pourrait ressembler une fois implémenté.
Enfin, nous avons commencé l'illustration d'objets du jeu (que l'on appelle aussi **sprites** d'objets).
La liste n'étant pas fixée à l'époque, nous n'avions aucune idée de la quantité d'illustrations nécessaire. Aujourd'hui, nous pouvons dire que nous avions fait plus de 35% des illustrations des objets totaux du jeu à ce moment-là.
Toutes ces illustrations ont pu être ajoutées dans le jeu grâce aux fonctionnalités du **canvas** de **Unity**. Elles ont contribuées à la mise en place de l'ambiance du jeu.
Vous retrouverez toutes ces illustrations en annexe.

4.7 Interface

Du côté de l'interface en jeu, nous avons décidé d'implémenter quelques fonctionnalités très tôt. En effet, cela nous a permis de découvrir comment marchait le système 2D de **Unity** avec ses nombreux **User Interface (UI)** (emplacements pour texte, images, boutons etc...).
Également, cela nous a permis de tester plus facilement certaines fonctionnalités du jeu telles que la vie, la récupération et l'utilisation d'objets.

4.7.1 Menu

Le menu fut un parfait terrain d'entraînement pour comprendre ces fonctionnalités.
En effet, lorsque nous arrivons sur le menu principal il y a quatre boutons. Nous avons lié ces derniers à des fonctions écrites dans un script "MenuManager".
Ces boutons permettent de naviguer entre les différents sous-menus, allant de la recherche d'une salle à sa création en passant par les options.
Dans les scripts nous faisons apparaître et disparaître les différents sous-menus en activant (ou désactivant) le **canvas** qui leur est lié.
Un des boutons présents dans le menu principal est "Create room" qui, comme son nom l'indique, va permettre à l'utilisateur de créer une salle en entrant le nom souhaité dans un **Input text**.
Une fois créée, son nom va apparaître dans le menu "Find Room" et un autre utilisateur pourra la rejoindre.
Finalement, les options nous ont permis d'en apprendre plus sur ce domaine.
La création des réglages sonores ont nécessité l'utilisation d'un **slider**. Ce dernier gère le volume de l'**AudioMixer** principal.
Et nous avons implémenté dans les réglages vidéos, la possibilité de se mettre en plein écran grâce à un **toggle**, et la possibilité de choisir la résolution du jeu par l'intermédiaire d'un **dropdown**.

4.7.2 Overlay

Du côté de l'**overlay** en jeu, il y a une barre de vie.
C'est en fait un **slider** dont l'apparence a été modifiée et qui varie en

fonction d'un script. La position du **slider** correspond au pourcentage de la vie actuelle du personnage par rapport à sa vie maximale.
De plus, un autre **overlay** apparaît lorsqu'on récupère un objet dans un coffre. Dans ce cas le nom, l'image et la description de l'objet vont s'afficher à l'écran pendant quelque secondes.
La difficulté de cette partie a été la récupération des informations. En effet, elles sont initialement stockées dans un dictionnaire d'objets. Nous avons donc dû modifier les paramètres des **TextContainers** et **ImageContainers** de l'**overlay** directement depuis un script.

4.8 Carte

Nous avons centré cette première phase de développement sur le multijoueurs, la jouabilité et la création d'un environnement de développement efficace. Cependant nous nous sommes rapidement retrouvés en avance sur ces différents domaines nous permettant ainsi de nous lancer plus rapidement sur la carte.
La première étape dans sa création a été l'ajout de nos premières salles.
Pour rappel la carte de **Below** est composée d'une multitude de salles reliées par des couloirs.
Grâce à notre avance nous avons eu le temps de créer presque la moitié des salles initialement prévues, la salle d'apparition, une grande salle à manger, une prison, une cuisine et enfin un laboratoire. Ces salles sont toutes sombres et reprennent un style médiéval.
Il a été nécessaire, pour la première soutenance de créer une version d'exposition de la carte en disposant chaque salle sur notre scène principale et en les reliant par des couloirs.

4.9 Modélisation

La modélisation 3D a été l'une des principales difficultés durant cette phase de développement car aucun de nous n'avait de compétences dans ce domaine.
Nous avons choisi le logiciel **Blender** pour la modélisation.
Ce logiciel a de nombreux avantages, surtout pour des personnes inexpérimentées telles que nous.
Premièrement, **Blender** est très facile à prendre en main même avec peu de temps d'utilisation. Deuxièmement, c'est un logiciel très connu

dans le domaine de la modélisation, ce qui permet d'accéder à de nombreux tutoriels et astuces sur Internet et ainsi de progresser rapidement.
Troisièmement, l'implémentation de modèles sur **Unity** est à première vue très facile. Ce logiciel étant utilisé par de nombreux épitéens, l'entraide est fréquente et très utile.
Blender est certes facile à prendre en main mais il est aussi très facile de s'y perdre.
Les fonctionnalités utiles sont éparpillées sur tout le tableau d'outils du logiciel. Nous avons eu quelques difficultés au début pour nous repérer. Mais avec l'aide de tutoriels, nous avons compris que 95% des fonctionnalités n'étaient pas utiles dans notre cas.

Il fallait être méthodique dans notre travail et séparer les modèles en sous-modèles pour les rassembler ensuite. Cela a permis un résultat plus propre et abouti.
Une problématique de la modélisation a été l'insertion des modèles terminés dans **Unity** afin d'en faire des objets préfabriqués.
En effet, les modèles ayant une texture demandent d'importer les composants séparément puis de les assembler à l'intérieur d'**Unity**.
Durant cette phase de développement, nous avons majoritairement pris en main **Blender** dans les grandes lignes.
Malgré cela, nous avons pu créer les premières versions des modèles du personnage jouable et d'un monstre à l'aide des illustrations en 2D.
Seul le monstre avait alors été implémenté dans le jeu car le personnage nécessitait un squelette d'animation qui était en cours de création.

4.10 IA

Du côté des intelligences artificielles, nous avions ajouté à **Below** notre premier ennemi : la "Lice" dont vous verrez le modèle en annexe. Pour ce faire nous avons dû, tout d'abord, ajouter un composant **NavMeshAgent** à l'objet du monstre.
Ce dernier indique, à l'intelligence artificielle à laquelle il est attaché, les différents endroits de la scène où elle peut se déplacer.
Une fois cela fait, nous avons créé trois comportements qui seront utilisés ou non selon la situation.

Tout d'abord il y a la patrouille, état dans lequel l'IA, tant qu'elle ne voit pas d'ennemi, va faire des rondes dans un périmètre donné depuis son point de départ.
Ensuite, si elle voit un joueur mais qu'il est trop loin pour l'attaquer, elle se déplace vers lui.
Enfin, quand elle est finalement à portée d'attaque elle va tirer des boules de feu semblables à celle du joueur.
Comme ce dernier, chaque ennemi possède un montant de vie qui lui est propre. Le joueur peut donc vaincre les ennemis en les attaquant.
Chaque intelligence artificielle est synchronisée en réseau par l'intermédiaire de Photon. Pour ce faire, il nous suffit d'instancier chaque ennemi à travers photon grâce à la fonction
PhotonNetwork.Instanciate() plutôt que la fonction **Instanciate()** de base de **Unity**.
Une fois cet objet préfabriqué instancié, il suit son comportement normal dans la scène du jeu.

5 Deuxième phase de développement

5.1 Corrections animations + mouvements

La deuxième phase de développement a commencé avec la correction de quelques problèmes de jouabilité constatés dans les premières versions du jeu. Nous avons tout d'abord corrigé les animations du joueur en retravaillant le module **animator** et en déclenchant certaines animations manuellement plutôt qu'à l'aide de transitions. Cela permet d'avoir un résultat final plus fluide et naturel.
Il a ensuite fallu retoucher les mouvements et la physique du personnage-joueur. Nous avons constaté que la différence de taille entre le **collider** et modèle créait un rendu peu esthétique (le personnage flottait légèrement au-dessus du sol). Nous avons corrigé cela en changeant la taille du **collider**.
Ensuite, nous nous sommes attaqués au saut. Il était jusqu'ici peu pratique à utiliser et cela était un vrai problème. Nous avons donc modifié les valeurs de vitesse de montée, de descente, de durée du saut etc... jusqu'à trouver un équilibre.

Une dernière étape fut l'ajout de particules sur les attaques du joueur et des ennemis. Pour cela, nous nous sommes procurés des particules de flamme dans l'**Asset Store** et les avons attachées au **GameObject** des projectiles.

5.2 Embellissement des menus

Comme dit précédemment nous n'avions jusqu'alors qu'un menu primaire permettant de lancer une partie et de modifier quelques options. Nous avons donc décidé d'y remédier en le refaisant complètement afin qu'il soit à la hauteur de nos attentes. Nous avons ainsi créé de nouveaux boutons, fonds d'écran et animations pour l'occasion.

5.2.1 Menu Principal

La première étape fut de recréer un squelette pour ce menu, nous nous sommes donc inspirés de l'ancien en l'étoffant un peu. Sachant que nous devions terminer le fond d'écran, nous avons utilisé son croquis en attendant, afin d'avoir une vision globale du résultat final.

Nous avons ajouté au menu principal un nouvel onglet contenant un bouton **New Game**, **options** et **Quit game**. Cliquer sur ce premier nous permet par la suite de choisir le mode de jeu (joueur seul ou multijoueurs).
Pour ce qui est du fonctionnement du menu, chaque onglet (ou sous-menu) correspond à un **GameObject** simple dont ses fils sont les boutons et autres **User Interfaces** contenus dans le sous-menu correspondants. Pour naviguer entre les différents sous-menus nous activons (ou désactivons) les **GameObjects** représentant les onglets, ce qui par la suite fait apparaître ou non ses fils.
Pour ce faire, nous donnons aux boutons des fonctions utilisant **SetActive(bool)** sur les différents sous-menus souhaités.

5.2.2 Menu En Jeu

Le menu en jeu a lui aussi été remis à neuf et amélioré dans ses aspects graphique et pratique.
En effet, le fond d'écran et les boutons ont eux aussi été remplacés pour que ce menu soit lui aussi cohérent avec le reste du jeu.
Du côté pratique, lorsque nous ouvrions ce menu la souris se débloquait et apparaissait bien. Cependant la caméra et les mouvements n'étaient pas bloqués rendant ainsi son utilisation assez désagréable. Nous avons donc remédié à cela en utilisant un booléen déjà créé qui indiquait si oui ou non le menu était ouvert. Lorsque ce dernier vaut **true** nous désactivons le **component Camera Freelook** du joueur rendant sa caméra statique. Ensuite, dans les scripts de déplacements et d'attaques nous avons ajouté le fait de **return** avant de faire quoi que ce soit si ce booléen est vrai.

5.2.3 Animations des Boutons

Comme dit plus haut nous avons ajouté aux nouveaux boutons des animations afin de rendre le menu plus vivant.
Ces animations se déclenchent selon l'état du bouton. Ce dernier jongle entre 3 modes que sont **Highlighted** lorsque nous passons la souris dessus, **Pressed** lorsque nous appuyons et **Normal** le reste du temps.
A l'état normal, comme son nom l'indique le bouton ne change pas, c'est son apparence de base.

Puis lorsqu'il est **Highlighted** la taille du bouton augmente et la couleur du texte change selon l'apparence du bouton.
Finalement, lorsque nous appuyons enfin dessus, la taille du bouton diminue puis ré-augmente pour donner un effet rebondissant. La couleur du texte change une nouvelle fois, un petit effet de particule apparaît, le **sprite** du bouton change et passe en mode appuyé.
Cependant ce petit effet de particule nous a obligés à refaire complètement les réglages de tous les **canevas** et caméras car sans cela, impossible de le voir.
Pour ce qui est de la particule en elle-même, nous voulions lui donner un effet d'étincelle. Pour ce faire, nous avons joué sur plusieurs paramètres tels que la gravité, la couleur selon son temps de vie, sa vitesse, sa taille et bien d'autres choses.

5.2.4 Partie graphisme

Du côté de la carte, nous avons continué de créer des salles à l'aide d'**asset** gratuit de donjon low poly. De plus, une salle au style antique a été ajoutée à l'intérieur du donjon. De ce fait, un nouvel **asset** de temple antique a été incorporé.

A l'instar de l'aspect pratique du menu, nous avons revu tout l'aspect graphique de celui-ci. Le menu étant la première chose que le joueur verra en lançant le jeu, l'aspect graphique est donc primordial. Nous avons tout d'abord créé quatre nouveaux boutons avec une version appuyée et non-appuyée pour chaque bouton. Enfin nous avons dessiné l'arrière-plan du jeu en prenant le temps de faire quelque chose de propre et détaillé.

5.3 Nouvelles Intelligences Artificielles

5.3.1 Modélisations et Squelettes

La modélisation a continué d'être l'une des principales difficultés durant cette période de développement car nous avons dû apprendre les fonctionnalités les plus dures de **Blender**.

5.3.2 Modélisation

Dans la continuité de la première soutenance, nous avons modélisé tout le bestiaire de **Below**, modifié certains modèles créés précédemment et ajouté quelques variantes pour les monstres d'élite.
Nous avons gardé les mêmes méthodes ce qui nous a permis d'être plus efficaces. Nous avons de plus, appris l'existence de certaines fonctionnalités permettant ainsi de modéliser avec plus d'assurance.

5.3.3 Squelette

La principale difficulté de la modélisation 3D a été d'implémenter les squelettes d'animations. Ceux-ci nous permettent d'animer correctement les modèles qui en possèdent. Nous avons donc appris à utiliser toutes les fonctionnalités en rapport avec ces squelettes.
Un squelette a été ajouté au personnage jouable mais aussi à la mimique et au bélial, deux monstres du jeu que vous pouvez retrouver dans l'annexe.
Nous avons eu de nombreuses difficultés quant à celui du personnage jouable et du bélial. Ceux-ci étaient très complexes ce qui rendait les erreurs fréquentes. Nous pouvons citer notamment des articulations non prises en compte par **Unity** ou des déformations de modèles.
Ces bugs ont été corrigés sur le personnage jouable d'où sa présence pour la deuxième soutenance. Cependant, ceux du bélial étant plus longs et difficiles à corriger, nous l'avons ajouté bien plus tard dans le jeu.

5.3.4 Comportements

De nombreuses IA ont été ajoutées durant la deuxième phase de développement. Afin d'en créer une grande variété dans des temps raisonnables nous avons profité des spécificités de la programmation orientée objet pour accélérer le processus. Les nouvelles IAs ont donc été créées comme des classes filles du premier ennemi.
Ainsi, nous avons écrit les comportements de nouveaux ennemis pardessus le modèle original afin de ne pas tout avoir à reprogrammer à chaque fois. Parmi les nouveaux monstres ajoutés il y a eu :
- les cafards très rapides et trop petits pour être touchés de loin mais n'ayant qu'un seul point de vie.

- les gouts qui ont beaucoup de points de vie et qui se divisent en deux lorsqu'ils meurent.
- les mimiques qui imitent l'apparence d'un coffre et attaquent le joueur lorsqu'il tente de l'ouvrir.

Nous avons également ajouté des ennemis dits "d'élite" qui sont des versions plus imposantes et plus puissantes des ennemis de base.
Ils ne peuvent être rencontrés qu'à des endroits spécifiques du jeu et laissent tomber un coffre à leur mort.

5.4 Nouvelles Salles

Durant la précédente phase de développement nous avons réalisé cinq salles, disposées sur une scène de démonstration. Pour cette seconde partie de développement, notre objectif était de réaliser une moitié de carte ainsi que la salle aux trésors. Nous avons donc réalisé les dernières salles manquantes : la salle ardente, la frontière et la salle aux trésors.

La salle ardente de forme allongée se trouve à proximité de la zone de départ. Elle a la particularité d'être entourée de lave et composée d'obstacles rendant sa traversée plus ardue. Cette difficulté de déplacement apporte un **gameplay** davantage centré sur la maîtrise des déplacements. Malgré sa dangerosité elle reste attrayante grâce aux deux coffres qu'elle renferme ; les autres n'en ont généralement qu'un.

La frontière est une grande salle permettant aux joueurs de rejoindre la salle aux trésors. Elle s'étend sur deux niveaux. Le premier est composé de deux petites salles comprenant un bureau et un rangement, ainsi que d'une grande salle ouverte permettant d'accéder à l'étage. Ce dernier permet d'entrer dans la salle aux trésors via un grand escalier et une porte s'ouvrant uniquement à un temps donné.

La salle aux trésors est la plus importante par sa taille et les objets qui s'y trouvent. Son rôle prépondérant est renforcé par son style graphique reprenant des temples et piliers antiques contrairement au reste de la carte qui est dans un style médiéval. Elle a aussi la particularité d'être située au centre de la carte et d'être la seule salle à

ciel ouvert.

La porte reliant la frontière à la salle aux trésors a nécessité l'ajout d'un script afin de s'ouvrir seulement après un certain temps. Une fois toutes les salles réalisées nous les avons disposées sur la scène où se déroule le jeu puis nous les avons reliées par des couloirs. L'ensemble ainsi formé représente la moitié de la carte.

5.4.1 Partie graphisme

Du côté de la carte, nous avons continué de créer des salles à l'aide d'**asset** gratuit de donjon low poly. De plus, une salle au style antique a été ajoutée à l'intérieur du donjon. De ce fait, un nouvel **asset** de temple antique a été incorporé.

5.5 Ajout d'objets

Du côté des objets, nous en avions implémentés plusieurs lors de la première partie, cependant beaucoup d'effets ne l'étaient pas. Beaucoup de fonctionnalités n'étaient pas encore disponibles ou du moins pas assez avancées pour pouvoir créer des objets les impactant.
Une fois le jeu plus complet, nous avons enfin pu ajouter plus de la moitié des objets prévus (c'est-à-dire 19).
Les objets restants n'ont pas été implémentés à ce moment-là pour différentes raisons, certains comme l'élixir de vie (qui permet d'avoir une vie supplémentaire) nécessitent la mise en place d'un système de mort. Aussi, certains n'étaient simplement pas encore imaginés.

5.5.1 Mécaniques

Il y a deux grands types d'objets, ceux augmentant une statistique, dans ce cas nous récupérons le script contenant la variable associée et nous la modifions.
Et ceux ayant un impact à un moment précis dans le jeu, ce qui complique un peu les choses. Pour ce genre d'objets nous avons créé de nombreux booléens dans différents scripts comme **CharacterThings** (qui comporte beaucoup de variables et de fonctions qui gèrent le

comportement du joueur) ou **NewShoot** (qui gère les tirs du joueur). Par exemple, lorsque le joueur prend des dégâts, les booléens vont être testés pour savoir si le joueur va disparaître momentanément ou si une partie des dégâts sera réduite.

Pour citer quelques objets il y a **L'Anneau Unique** qui permet de passer inaperçu aux yeux des ennemis. Pour le mettre en place, nous avons fait en sorte que les joueurs qui ont leur booléen **ring** vrai ne soient pas pris en compte dans la recherche des ennemis.

Autre exemple intéressant, la cape qui permet de devenir invisible pendant un court instant après avoir reçu un coup. Pour mettre en place son effet, nous passons tout d'abord le booléen **cape** du joueur à vrai. Ainsi, lors de l'utilisation de la fonction **TakeDamage** cela va désactiver l'apparence du joueur pendant deux secondes.
Avant d'implémenter la nouvelle apparence du joueur, nous récupérions le booléen du **Renderer** du joueur avec **GetComponentInChildren<SkinnedMeshRenderer>().enabled** et nous le passions à **false**.
Quand son apparence est devenue un puzzle de pièces d'armure, nous avons rassembler tous les composants visuels en un seul **GameObject** que nous désactivons au moment voulu afin de tous les faire disparaître d'un coup.
Finalement, il y a le **Vampirisme** qui permet de regagner quelques points de vie après avoir tué un monstre. Donc, comme d'habitude en récupérant cet objet, le booléen **vampire** du joueur passe à **true**.
Puis lorsqu'un monstre prend des dégâts et si le **GameObject "owner"** à l'origine de ces derniers est un **vampire**, nous vérifions si ces dommages sont létaux. Nous vérifions donc si le **GameObject** du monstre est encore présent dans le jeu en testant son égalité avec **null**. Si l'égalité retourne **true** cela veut dire que le monstre a été détruit, et nous ajoutons donc 10 points de vie au joueur.

5.5.2 Illustration

Certains autres aspects graphiques ont progressé durant cette période.
Tout d'abord le bestiaire a été intégralement dessiné, ce qui est notamment utile pour le site internet mais aussi pour la modélisation.
En parallèle, nous avons continué de faire des **sprites** d'items direc-

tement utilisés dans le jeu.
Enfin, nous avons fait un icône du jeu que vous pourrez retrouver en annexe.

5.6 Création du site Web

Comme indiqué dans le cahier des charges, nous avons commencé le développement du site internet durant cette phase. Pour cette première étape nous avions prévu la création d'une page d'accueil, d'une **timeline** ainsi que la mise en ligne du site internet.

La page d'accueil a été réalisée en **HTML** et **CSS**. On y retrouve des informations concernant le jeu comme des captures d'écran ou la présentation de certains objets. En haut du site, on retrouve un bandeau permettant la navigation entre les différentes rubriques, notamment la **timeline**. Cette dernière a elle aussi été faite en **HTML**, **CSS** mais a également nécessité l'utilisation du **JavaScript**. La **timeline** retrace les étapes clés de l'avancement de **Below** et permet de voir son état actuel. On y retrouve aussi les futures étapes de son développement. Comme indiqué précédemment, la **timeline** a nécessité l'utilisation du **JavaScript**. Ce dernier a été utilisé afin de créer une animation faisant apparaître chaque étape lors du défilement vers le bas de la page. En effet, chaque étape apparaît seulement lorsque la barre de défilement se trouve à son niveau.

5.7 Quelques difficultés...

Peu de temps avant la deuxième soutenance, lorsque nous répétions la présentation, plusieurs problèmes ont été découverts. Des **Room** n'apparaissaient pas et certains ennemis avaient un comportement étrange pour une raison inconnue. Nous avons dû régler ces problèmes rapidement ce qui ne nous a déstabilisé car nous craignions de les retrouver le jour J. Nous ne le savions pas à ce moment-là, mais ces problèmes allaient être d'une importance capitale pour notre multijoueurs.

6 Troisième phase de développement

6.1 Prise de conscience

Une fois la deuxième soutenance passée nous nous sommes mis en tête de régler quelques problèmes dont nous avions la connaissance.
En cherchant un peu leur origine nous nous sommes rendus compte que ces problèmes n'étaient en fait pas des cas isolés. De nombreuses fonctionnalités n'étaient pas bien synchronisées et nous ne comprenions pas pourquoi.

C'est alors qu'en fouillant dans la documentation de **Photon** et les différents forums, nous avons découvert les **Remote Procedure Calls** ou plus communément appelé **RPC**. Les RPC sont des fonctions de **Photon** qui permettent d'utiliser n'importe quelle autre fonction donnée en paramètre. Cependant l'intérêt de cette méthode réside dans le fait que nous pouvons indiquer sur quels PC elle se déclenche.
Par exemple, un joueur tire sur un monstre. Si nous utilisons la fonction **TakeDamage** sur celui-ci par l'intermédiaire d'un RPC et que nous décidons de la déclencher chez tous les joueurs (avec le paramètre **RpcTarget.All**), alors la vie du monstre va descendre chez tout le monde. Avant ces modifications, la vie du monstre n'aurait baissé que dans le jeu de celui qui a appliqué les dégâts. Les RPC permettent donc de mettre en commun les différentes actions faites par les joueurs.

6.2 Modification réseau

Après avoir découvert cette fonctionnalité il fallait non seulement modifier de nombreuses fonctions ainsi que leur structure.
En effet, nous ne pouvons pas passer en paramètre des variables lourdes comme des **GameObject**. Nous avons ainsi dû passer par la **PhotonView** de ces derniers.
Pour ce faire, nous récupérons le numéro identité de cette dernière avec **GetComponent<PhotonView>().viewID**, nous le passons en paramètre. Et une fois dans la fonction, nous récupérons la **PhotonView** et son **GameObjet** associé avec la méthode **PhotonView.Find(viewID).gameObject**.

Les fonctions passées en RPC sont nombreuses mais les principales sont les différents **TakeDamage** des joueurs et monstres, ainsi que de nouvelles fonctions comme **Heal**, qui comme son nom l'indique permet de signifier à tout le monde qu'un joueur s'est soigné. Et surtout le système d'instanciation des coffres a complètement été refait car le contenu de ces derniers n'était pas synchronisé.

Pour remédier à ce problème, au lieu de placer directement les coffres sur la carte nous plaçons un objet préfabriqué contenant un script qui va faire en sorte que :
- seul le maître de la salle peut instancier le coffre (afin qu'il n'y en ait qu'un).
- une fonction RPC **Start_chest** est appelée avec comme paramètres **RpcTarget.all** et des nombres aléatoires afin d'initialiser le coffre de la même façon pour tout le monde.

Finalement, la dernière fonction importante qui a été passée en RPC est la fonction **OpeningChest**, utilisée comme son nom l'indique lors de l'ouverture d'un coffre. Cela va donc permettre de synchroniser à la fois l'état et l'animation du coffre ainsi que l'inventaire des joueurs.

6.3 Implémentation du son

6.3.1 Mise en place

Jusqu'ici l'ambiance musicale de **Below** était quasi absente car elle se composait exclusivement d'un petit son qui se déclenchait lors de la mort d'un monstre. Nous l'avions implémenté simplement pour ne pas laisser un terrain inexploré.
Alors que le son paraissait moins important que le reste, nous voulions qu'il soit à la hauteur afin de rendre l'expérience de jeu encore plus immersive.

Ainsi, il a fallu tout d'abord mettre en place un **AudioMixer**, cet outil permet de gérer tout ce qui est en rapport avec le son sur un projet **Unity**.
La deuxième étape a été de créer trois **AudioMixerGroup** que sont **Master**, **Effect** et **Music**. Ils nous ont permis de répartir les types de son selon leur utilité pour une meilleure gestion.

Ainsi les musiques passent à travers l'**AudioMixerGroup Music**, les différents sons comme les tirs ou les morts passent à travers **Effect** et finalement ces deux **AudioMixerGroups** passent à travers **Master**.

Procéder de la sorte nous permet notamment de gérer séparément le volume de ces trois groupes. Si la musique est trop forte nous baissons le volume du groupe **Music**, idem pour les effets et si nous voulons baisser le volume du son en général nous baissons le volume de **Master**.

Une fois tout ceci mis en place nous devions implémenter les sons eux-mêmes. Lorsque nous voulions jouer un son, nous le liions à un script et l'activions depuis ce dernier.
Cependant cette méthode s'est révélée peu appropriée car nous ne pouvions ni gérer les paramètres de ce dernier correctement ni jouer un son en boucle proprement.
Ainsi nous avons, pour chaque son émis par un **GameObject**, ajouté un **component AudioSource**. Ce dernier permet de modifier les paramètres d'un son, de lui attribuer un **AudioMixerGroup** ou de pouvoir le faire jouer en boucle (très pratique pour les musiques notamment).
Une autre fonctionnalité très importante de ce composant est le fait de pouvoir gérer les sons en 3D et leur diffusion dans l'espace. Car sans cela nous entendrions tous les bruits du jeu à n'importe quel endroit de la carte.

6.3.2 Ajout de bruitages et musiques

Durant cette phase finale de travail nous avons ajouté de nombreux sons et musiques un peu partout. Tout d'abord, dans le menu principal nous avons choisi une musique relativement calme et joyeuse avec une ambiance médiévale. En effet, nous voulions montrer au joueur qu'il est encore dans une zone sans danger.

A l'inverse, une fois dans le donjon, la musique change complètement d'ambiance et devient sombre et oppressante. Nous l'avons ainsi choisie pour signifier au joueur que les choses sérieuses commencent et que les ennuis arrivent.

Pour ce qui est du reste des effets sonores nous voulions que le jeu soit

vraiment vivant. Nous avons donc essayé d'implémenter un maximum de bruits d'ambiances un peu partout comme le feu dans les cheminées, les torches et autres objets enflammés, mais aussi lorsque les portes et les coffres s'ouvrent, lorsque des boules de feu sont tirées ou lorsque le joueur prend des dégâts.

Finalement les monstres ont tous au moins un son joué à leur mort et certains jouent un son de manière constante comme le cafard par exemple qui émet un son d'insectes se déplaçant pour le plus grand plaisir des entomophobes.

6.4 Cartes finales

6.4.1 Finalisation de la carte multijoueurs

A la fin de la dernière phase de développement nous disposions d'une moitié de carte et de la salle aux trésors. Nous avons donc simplement placé la seconde moitié de carte de l'autre côté de la salle aux trésors. Les deux parties de la carte sont donc reliées par la salle aux trésors mais aussi par deux autres couloirs. Le premier est situé en haut de la carte, le second en bas et la salle aux trésors fait office de passage central.

Une fois la carte multijoueurs terminée, nous avions simplement une carte avec des portes qui s'ouvrent à un temps donné. Il nous a donc fallu ajouter plusieurs fonctionnalités de **Gameplay** tel qu'une mécanique de mort et de résurrection.
Cet ajout a nécessité l'implémentation d'une salle mortuaire où le joueur serait téléporté à sa mort. Cependant la mort dans **Below** n'est pas définitive avant 8 minutes. Nous avons donc ajouté un script qui téléporte les joueurs à leur zone d'apparition et leur redonne tous leurs points de vie.
Une autre fonctionnalité importante et liée à la carte a été l'ajout d'équipes. En effet dans les versions précédentes du jeu tous les joueurs apparaissaient au même endroit. Pour régler ce problème les joueurs doivent choisir une équipe lorsqu'ils rejoignent la **Room**.
Ce choix les fait apparaître dans la partie de carte correspondant à leur équipe. Pour implémenter ce système d'équipes nous avons utilisé la classe **Enum** du C#.

6.5 Mode solo

Une fois le mode multijoueurs terminé, il a fallu nous atteler à la création d'un mode un joueur. Celui-ci consiste en un parcours différent des environnements déjà existants.
Le joueur devra ainsi parcourir les grandes salles du donjon afin d'y trouver un maximum d'objets qui le rendront plus fort pour affronter le boss.
Les environnements sont particulièrement sombres, les coffres difficiles à trouver et les ennemis difficiles à repérer.
Le mode solo se conclut sur l'affrontement final contre le boss ; ultime difficulté avant la victoire.
Le jeu est globalement plus difficile qu'en mode multijoueurs afin de compenser l'absence d'adversaire humain.
Mourir dans ce mode de jeu est définitif et le joueur devra alors recommencer une nouvelle partie. Cela permet de créer une mécanique de **Die & Retry** qui rend la victoire finale réellement satisfaisante.

L'écran de victoire du mode solo informe le joueur du temps qu'il lui a fallu pour terminer sa partie, l'encourageant ainsi à battre son propre record en recommençant.
Il est en effet possible d'optimiser son parcours par la connaissance des environnements, des ennemis et des objets. Cela permet l'existence d'un défi consistant à essayer de terminer le mode solo le plus rapidement possible.
Un temps d'une partie "normale" serait entre 15 et 20 minutes alors que le temps d'une partie optimisée peut passer sous la barre des 4 minutes !

6.5.1 Carte solo

Pour la création du mode **un joueur** (sur lequel nous reviendrons plus tard), nous avions besoin de créer une carte adaptée. Ainsi, nous avons réutilisé les salles déjà créées pour le mode multijoueurs que nous avons modifiées afin de les agrandir. Elles ont ensuite été disposées les unes à la suite des autres afin de créer un effet de long couloir.
Tout cela a été fait afin de rendre la carte du solo plus longue à parcourir et obliger le joueur à passer par toutes les salles.
La disposition des coffres et des monstres dans les salles est égale-

ment différente de celle du mode multijoueurs. Ils sont plus nombreux afin de compenser l'absence du second joueur pour que la tension du jeu reste aussi intense.

Nous avons ajouté de nombreux éléments interactifs comme des objets pourvus de **RigidBody** afin que le joueur puisse les déplacer. On y trouve également des chaudrons qui permettent de régénérer les points de vie du joueur et un portail permettant de rejoindre le boss à la fin du jeu (nous y reviendrons également).

6.6 Nouveaux Graphisme

L'objectif pour cette période finale a été de peaufiner l'aspect graphique du jeu.

6.6.1 Finalisation graphique de la carte multijoueurs

Dans un premier temps, nous avons uni les différentes parties de la carte.
Dans un second temps chaque erreur d'affichage et chaque trou dans les murs ont été corrigés pour rendre la carte plus belle.
Enfin, la finalisation de la carte s'est faite par l'ajout de salles d'attente après une mort, celles-ci sont séparées du reste de la carte. Chaque joueur a sa propre salle de mort car les deux joueurs ne doivent pas interagir entre eux avant de ressusciter.

6.6.2 Écrans de chargement et de mort

Tout d'abord, nous avons dessiné un écran de chargement à la manière de l'écran d'accueil.
Celui-ci est cependant bien plus sombre et laisse entrevoir la silhouette du boss du mode solo ainsi que ses sbires.
Puis, l'écran de mort définitive a été implémenté. Celui-ci, contrairement à l'écran de chargement, n'est pas une illustration mais un simple écran noir avec un texte rouge expliquant au joueur qu'il est mort.
Cet écran, plus simple que les autres, est une référence à de nombreux jeux ayant le même procédé, notamment **Dark Souls**.

6.6.3 Objets et illustrations

Les derniers **sprites** d'objets ont été dessinés durant cette période de développement.
Ceux-ci ont été finis en avance pour pouvoir passer à d'autres tâches toutes aussi importantes. Le total d'objets dans le jeu final s'élève donc à 35.
Les **sprites** d'objets sont tous très différents les uns des autres. Cela permet au joueur d'identifier l'objet rapidement ainsi que son effet. De plus, les **sprites** de rareté qui sont derrière l'objet lors de leur récupération, donnent une idée au joueur de la puissance de l'objet.

Pour finir, nous avons réalisé des **Artworks** afin d'illustrer les documents liés aux jeux (site internet, livrets, rapports, etc). Nous avons aussi modifié quelques anciennes illustrations pour les rendre plus présentables, sur le site notamment.

6.6.4 Finalisation de la modélisation

L'objectif final pour la modélisation était de finir le squelette du bélial, un monstre du jeu. Malheureusement sa finalisation a pris plus de temps que prévu car les bugs d'articulation que nous avions précédemment persistaient.
De plus, les bugs étant sur le logiciel **Blender**, nous n'avions aucun moyen de savoir précisément d'où venaient l'erreur. Nous avons donc été obligés de refaire plusieurs fois le modèle entier du bélial afin que le squelette fonctionne.
Nous avons aussi corrigé certains défauts des anciens modèles avec notamment des textures inversées, quelques petits trous dans les modèles et quelques erreurs de rotations.

6.6.5 Refonte graphique

Nous avons fini nos objectifs de graphisme en avance. Cela nous a donc permis de pouvoir revoir certains aspects graphiques de notre jeu et les modifier.
Tout d'abord les particules dans le jeu ont été changées pour les rendre plus jolies. Nous pouvons citer les boules de feu, les projectiles ou encore l'effet de lumière sur le bélial.
De plus, nous avons utilisé du **post-processing** pour l'étalonnage du

jeu afin de le rendre plus beau.
D'autre part, nos logos qui ont été faits durant la genèse du projet ont été revus.
Nous avons donc refait le logo **MTW** pour le rendre plus sobre et le logo **Below** pour qu'il soit cohérent avec l'icône du jeu.
Enfin, comme nous l'avons dit précédemment, nous avons modifié les couleurs de quelques illustrations de monstres après des retours avisés d'observateurs.

6.6.6 Post-processing

Afin d'améliorer le rendu visuel de **Below**, nous avons ajouté un système de **post-processing** dans le jeu.
Celui-ci demande l'installation du **package Unity** du même nom ainsi que de certaines connaissances en étalonnage. Nous avons ajouté un **gameObject Post-process Volume** à la caméra du **prefab** du personnage jouable.
Cela permet d'appliquer un certain nombre d'effets à l'image capturée par la caméra et renvoyée au joueur. Les effets que nous avons ajoutés sont :
- Le **Motion Blur**, créant un effet de flou de mouvement.
- Le **Bloom**, qui permet d'accentuer fortement l'intensité et la diffusion de la lumière.
- Le **Color Grading**, permettant de faire ressortir certaines teintes par rapport à d'autres (dans notre cas le filtre **ACES**).

Tout cela permet d'accentuer l'ambiance pesante du jeu et d'améliorer l'immersion.

6.7 Dernières IAs

La dernière intelligence artificielle ajoutée au jeu fut celle du bélial (voir illustration et modèle en annexe). Bien que ce monstre soit impressionnant dans le jeu, il se contente de surcharger les scripts d'autres ennemis dans le code.
Il possède en effet les capacités combinées de plusieurs autres monstres du jeu en plus dangereuses :
- Il patrouille et tire comme la Lice mais sur une plus grande distance et possède des projectiles plus rapides et puissants.

- Il attaque au corps-à-corps comme un cafard mais inflige plus de dégâts.
- Il possède également davantage de points de vie que le reste du bestiaire.

Le bélial représente ainsi une menace plus importante que les autres ennemis du jeu et devra souvent être fuit s'il est rencontré en début de partie.

Il existe une version "élite" de ce monstre qui a la particularité de se séparer en deux entités différentes lorsqu'il tombe sous la moitié de ses points de vie : la tête et le corps.
En pratique, lorsque la variable **HP** de son script passe sous la moitié des **HPmax**, le **MeshRenderer** de sa tête est désactivée.
Un nouveau **gameObject** (sa tête) est ensuite instancié devant lui en tant que nouvel ennemi.
Lorsque l'une des deux entités meurt, elle vérifie si l'autre l'est également et si tel est les cas, elle fera apparaître un coffre.
Il est également à noter que le Bélial est le seul ennemi du jeu (boss mis à part) à posséder un squelette et des animations. Ces dernières fonctionnent comme celles du joueur : elles sont déclenchées par un script selon l'état du **gameObject** (ne bouge pas, se déplace, attaque, etc).

Une fois le Bélial terminé, il a fallu s'atteler à la création d'un boss pour le mode solo. Nous voulions conclure notre jeu sur un moment fort, l'intelligence artificielle du boss a donc demandé plus de temps et de réflexion que celles des autres ennemis.
En effet, le boss peut interagir avec l'environnement et a des mécaniques uniques dans le jeu.
Nous avons donc du recréer un nouveau comportement à partir de zéro plutôt que de surcharger les méthodes des autres monstres du jeu.
Le boss est indissociable de la salle dans laquelle il se trouve, une étroite plateforme flottant au-dessus d'une mare de lave. Il faudra ainsi faire attention à la chute pendant ce long combat. Ce dernier se déroule en quatre parties que voici :

Partie 1 : Le joueur doit sauter sur des piliers au-dessus de la lave pour rejoindre la plateforme du boss, en esquivant les tirs de ce der-

nier.

Partie 2 : Le boss renvoie les piliers dans la lave et se met à charger le joueur qui est désormais piégé sur la plateforme. Ce dernier aura la possibilité de se cacher derrière des rochers pour se protéger des attaques.

Partie 3 : Des piliers apparaissent aux quatre coins de la plateforme et commencent à faire apparaître des gouttes de sang. Le joueur devra alors détruire ces dernières avant qu'elles n'atteignent le boss car ce dernier les utilise pour se soigner.

Partie 4 : Le boss devient invincible et la lave commence à monter. Des piliers sortent à nouveau de la lave et permettent au joueur d'atteindre un orbe plus loin. Activer cet orbe permet de tuer le boss et de mettre fin au combat.

Cet affrontement avec le boss fut particulièrement complexe à mettre en place car il a fallu penser à des interactions avec l'environnement. L'écriture de nombreux scripts a également été nécessaire (ceux des gouttes de sang par exemple).

6.8 Autres ajouts

Quand toutes les fonctionnalités prévues ont été implémentées, nous avons pris la décision d'utiliser notre temps restant pour en ajouter de nouvelles.
Nous avons ainsi ajouté une torche au joueur afin d'éclairer les endroits trop sombres. Il suffit ainsi d'appuyer sur la touche F afin de sortir une torche.
Utiliser cette dernière empêche néanmoins le joueur d'attaquer, il faut donc perpétuellement choisir entre la visibilité et l'attaque.
Nous avons ajouté un nouvel effet, l'empoisonnement causé par certains ennemis ou objets qui inflige lentement des dégâts au joueur jusqu'à la mort, créant ainsi un contre-la-montre supplémentaire.
Un chronomètre est désormais présent sous la barre de vie afin que le joueur garde un oeil sur le temps restant.
Quelques correctifs sur la caméra ont également été effectués afin que cette dernière soit plus agréable. La distance de vue à été dimi-

nuée afin de limiter la vision du joueur.
L'angle de vue a été reculé pour permettre une meilleure apprécia-tion des distances.

Une fois tout cela terminé, il nous a fallu tester de très nombreuses fois le jeu afin d'y trouver les bugs et les corriger. Ce travail fut long et fastidieux mais nécessaire afin que le jeu soit parfaitement jouable et agréable.

6.9 Finitions du site web

Pour rappel le développement du site web a commencé après la première soutenance et nous disposions d'une page d'accueil et d'une **Timeline** après la deuxième.

6.9.1 TimeLine

Nous avons mis à jour la **Timeline** pour qu'elle suive l'avancement du projet.
Nous avons ajouté les rubriques manquantes.
En effet chaque étape contenait un bouton « Lire plus » qui n'était pas encore opérationnel.
Maintenant, chacun d'eux renvoie vers une page décrivant l'étape de développement en question. Cette description contient une ou deux images ainsi qu'un texte à propos des moyens mis en place pour réaliser l'étape.

6.9.2 Galerie

L'une des particularités de **Below** est son système de récupération d'objets. Au vu du grand nombre d'objets disponibles et de leurs effets en tout genre, il nous a paru nécessaire de regrouper tous les objets dans une galerie. Vous y retrouverez donc un descriptif de chaque objet, leur rareté ainsi que leur histoire. La galerie contient aussi des informations sur les monstres.
Pour réaliser la galerie nous avons utilisé la fonctionnalité **wrap** du **CSS** permettant d'aligner simplement différents **items**. Chaque **item** dispose d'un bouton « Découvrir » amenant à la page descriptive de

celui-ci.

6.9.3 Téléchargement

L'une des fonctionnalités requises pour notre site web était le téléchargement de **Below**.
Dans ce but, nous avons créé une toute nouvelle rubrique intitulée *Téléchargement* directement accessible depuis la barre de navigation du site.
Cette page contient un bouton de téléchargement qui envoie à la **Release** du jeu sur son répertoire **GitHub**.

6.9.4 MTW

La dernière rubrique ajoutée à notre site est une page décrivant notre groupe et sa formation. Elle indique aussi le rôle qu'a joué chaque membre dans le développement de **Below**. Cette rubrique est également accessible depuis la barre de navigation.

6.10 Installateur

Pour rendre l'installation du jeu plus simple pour les joueurs, nous avons décidé d'utiliser un installateur et plus particulièrement **Inno Setup**. Ce dernier est simple d'utilisation pour le client et nous permet d'ajouter une multitude de fichiers supplémentaires comme une licence ou un manuel d'installation. Il permet aussi à l'utilisateur de choisir l'emplacement d'installation du jeu.
Il compresse le jeu rendant son téléchargement bien plus rapide.

De notre côté la configuration d'**Inno Setup** a été rapide. Nous avons simplement ajouté les informations sur notre jeu ainsi que ses fichiers dans une architecture précise.

6.11 Bande annonce

Pour réaliser la bande annonce nous avons utilisé comme logiciel de montage **Filmora X** développé par **Wondershare**.
Les différents plans la composant ont été directement pris dans **Unity** en utilisant l'**Animator**. Nous avons simplement créé des animations

et nous les avons faites jouer à notre caméra ou à tout objet que nous voulions déplacer durant le plan.

Notre bande annonce a pour but de faire découvrir l'univers de **Below** et non son **Gameplay**. C'est pour cela que chacun de ses plans présente un nouveau décor ou monstre de la carte.

6.12 Boite du jeu

Pour la boite du jeu, nous avons décidé de faire quelque chose d'original pour tenter de nous démarquer.

6.12.1 Jaquette de jeu

Nous avons choisi pour la jaquette de jeu le format **Blu-Ray** utilisé par la majorité des boites jeu vidéo.
De plus, **The Elder Scrolls V : Skyrim** a été une inspiration lors de la création de la jaquette de **Below**. Celle-ci présente simplement le logo du jeu sur un fond de pierre noire.
Nous avons voulu nous inspirer de cette idée de conception car elle présente d'une manière simple, distincte et claire le jeu. Ainsi l'avant de notre jaquette présente l'icône de **Below** entouré de dessins runiques gravés dans une roche violette. Sur l'arrière de la jaquette nous avons décrit brièvement le jeu en montrant quelques images.
Vous la retrouverez en annexe.

6.12.2 Accessoires et décorations

A la manière de certaines éditions limitées de jeux vidéo, nous voulions avoir une présentation plus étoffée qu'une simple jaquette.
Ainsi, nous avons trouvé une boîte en bois rappelant un grimoire faisant écho à l'univers du jeu. Dans cette boîte se trouve, en plus de la jaquette, un manuel d'installation, un manuel de jeu et une carte du donjon de **Below**.
Le jeu, quant à lui, se trouve sur une clef USB, elle-même dans la jaquette du jeu.

7 Évolutions de Below dans le futur

Bien que la durée prévue de développement de **Below** soit écoulée et que nous ayons implémenté tous les éléments que nous souhaitions, il n'est pas exclu que nous continuions à enrichir le jeu dans le futur.
Cela se fera probablement par l'ajout de nouveaux objets aux effets uniques et de nouveaux ennemis afin d'augmenter la diversité des situations.
On peut également penser à l'ajout de nouveaux environnements pour le mode solo afin d'augmenter sa durée de vie et proposer de nouvelles difficultés.
Un ajout plus complexe et ambitieux serait la génération procédurale des environnements. Lors d'une partie, une nouvelle carte serait générée par l'ordinateur à partir des décors déjà existants.
Cela demande néanmoins énormément de temps que nous n'avons pas pour le moment.
Tous ces ajouts seraient permis par la flexibilité du genre du **Rogue Lite**, qui rend l'ajout de nouveaux éléments ou mécaniques simple et fluide.

Enfin, nous pourrions agrémenter le jeu de nouveaux fonds d'écran uniques. Mais la création de ces derniers prend du temps c'est pourquoi nous n'en avons que deux dans le jeu actuellement.

8 Bilan du projet

8.1 Bilan général de l'équipe

Globalement, le résultat final étant très proche de nos attentes de départ, nous sommes fiers de ce que nous avons produit et du chemin parcouru.
Nous sommes très contents d'avoir su gérer tant bien que mal les nombreux problèmes et difficultés rencontrées.
De plus l'aboutissement de ce projet clôt notre première année à EPITA et finalise notre première réalisation de grande envergure.
Nous avons trouvé que nos méthodes de travail étaient plutôt efficaces car chaque membre du groupe était à la fois motivé et prêt à donner le meilleur de lui-même.
De ce fait, l'ambiance de travail était excellente ce qui a rendu ce travail à la fois enrichissant et plaisant.