Rapport de projet

Belloni Corentin Blancheton Tom
Cizeron Lucas Geoffray Flavien

Mardi 7 Juin à 14h

Table des matières

1 Introduction

Ce rapport de projet retrace les différents éléments ajouté au projet durant ces 6 derniers mois. Il détaille l'implémentation de chaque partie du jeu mais aussi des problèmes rencontrés, les solutions trouvées et les décision prise durant son développement.

De ses débuts jusqu'à aujourd'hui, l'Histoire de Jeu vidéo à été marqué par énormément jeux. De toute évidence, le jeu de Zombie fait partis des jeux qui ont marqué des générations de joueurs et continue de les marquer encore de nos jours. Le jeu de zombie est aussi l'un des plus « bankable », qui n'a jamais rêvé de se défouler sur des créatures.

Marqué et inspiré par des jeux comme Resident Evil, the Last of Us, ou encore Call of Duty : Zombie, l'équipe du Projet Zakharov présente Zahkarov : Undead Prison. Un shooter de zombies arcade qui rend hommage à toutes ces œuvres littéraires et cinématographique et tout ces jeux qui ont fait du zombie une créature mythique de l'Histoire du jeu vidéo.

Zakharov : Undead Prison reprend les codes des jeux de survie et de zombies. S'équiper et courir sont vaut seules chances de survis. Le joueur à pour but de survivre le plus longtemps possible et d'éliminer un maximum de zombies.

Tout d'abord, un récapitulatif du cahier des charges sera fait qui présente l'histoire du jeu, son objectif ainsi que le groupe derrière son développement. De plus, nous avons un tableau de la répartition des tâches pour chaque membre du groupe et un tableau de l'évolution du jeu à chaque soutenance.

Ensuite, ce rapport expose l'avancement du projet dans plusieurs grandes parties qui représentent les fondations de notre jeu. Le plan est organisé comme une frise chronologique de chaque étape rencontrée par le joueur. En effet, le menu représente la première interaction que le joueur a avec le jeu. Ensuite, il découvre l'environnement du jeu représenté par la carte et les effets lumineux. Puis le joueur comprend le gameplay et prend le jeu en mains en faisant la rencontre avec les ennemies et les armes, partie essentielle du jeu. Pour plus de plaisir, il peut jouer avec des amis.

Enfin, chaque personne du groupe détaille son bilan personnels de son expérience sur la réalisation du projet.

2 Résumé du cahier des charges

2.1 Histoire

Le jeu prend place dans une prison désaffectée de l'URSS en pleine Seconde Guerre Mondiale, dans le Nord de la Sibérie. Les joueurs se réveillent dans une prison sombre, délabré et vide au premier abord.

En explorant cette prison, les joueurs se rendent compte que les lieux sont loin d'être inoccupés. Tous les prisonniers et les gardes sont infectés et veulent leur mort. Malheureusement pour eux, quelque chose de plus gros encore, semble leur vouloir du mal.

Les joueurs n'ont alors qu'un seul but, survivre pour s'échapper. Par chance ils trouves de vieille armes abandonnée appartenant sûrement aux anciens gardes de cette mystérieuse prison. Ils ont finalement peut être une chance de survivre...

2.2 Objectif

L'objectif du jeu est donc de survivre à chaque tour d'invasion de zombies, après quoi un tour plus difficile commence.

Le joueur commence le jeu avec une arme peu efficace. En combattant les zombies, les joueurs reçoivent des points appelé des ZCoins. Ces points sont utilisables pour acheter des armes répartis dans la prison ou ouvrir des portes afin d'évoluer dans le jeu.

En fonction des manches, des variations de zombies apparaissent lors de vagues spécifiques. Ces zombies spécifiques pourront par exemple avoir une capacité à toucher les joueurs de plus loin, de l'immobiliser pendant un certain temps, etc...

Les zombies entrent depuis l'extérieur de la carte jouable par des points d'apparition non accessible par le joueur. A leur mort, les zombies font tomber des ZCoins d'une valeur différente selon le type de zombie.

Le joueur est capable de courir pour échapper aux hordes de zombies.

2.3 Le groupe

2.3.1 Belloni Corentin

Je m'appelle Corentin, j'ai 20 ans. C'est à partir de mes 10 ans que j'ai commencé à découvrir le monde du jeu vidéo et de l'informatique en général. Depuis ce jour-là, j'ai su que je voulais en faire mon métier. J'ai joué à tous types de jeux : bac à sable comme Minecraft, MOBA et évidemment au shooter comme Call of Duty (en particulier à son mode Zombie). C'est suite à ces nombreuses heures de jeux que j'ai développés l'envie de créer mon propre jeu vidéo.

2.3.2 Blancheton Tom

Je m'appelle Tom et j'ai 18 ans. Je joue aux jeux vidéo depuis tout petit et par conséquent j'ai pu en tester un grand nombre. Le type de jeux qui m'a le plus marqué est les FPS en multijoueur. Notamment Paladins, Minecraft, Fortnite et Call Of Duty. Il me semblait donc logique de vouloir faire un jeu de ce type !

2.3.3 Cizeron Lucas

Je n'ai jamais trouvé suffisant de jouer uniquement aux jeux vidéo, c'est pourquoi je me suis toujours penché vers des jeux incluant des éditeurs de map ou tout autre outil permettant de créer ma propre expérience. Dans cette même optique, j'ai beaucoup utilisé Unreal Engine et unity étant plus jeune mais sans savoir programmé, ce qui limitait beaucoup les possibilités. Ce projet est l'occasion parfaite de pouvoir créer en ayant cette fois des bases en programmation.

2.3.4 Geoffray Flavien

J'ai 18 ans et je joue aux jeux vidéo depuis que je suis tout petit. J'ai pu jouer à beaucoup de jeux mais ceux qui m'ont le plus marqué et auxquels j'ai le plus joué sont certainement les shooter tel que Destiny ou Call of Duty. Il était donc naturel pour moi d'avoir envie de participer à la création d'un de ces jeux. Je suis très enthousiaste à l'idée de me lancer dans sa conception et de le rendre concret.

2.4 Répartition des tâches

Voici le tableau récapitulatif de qui à fait quoi dans ce projet.

Légende :
°R représente le responsable (la personne qui gère principalement cette partie).
°S représente les suppléants (les personnes qui ont contribué).

Nom / Tâches	Corentin	Tom	Flavien	Lucas
Interface		R		S
Logo		R		S
Son	S		S	R
Gameplay	R	R	R	R
Game Design	S		S	R
Animation		S	R	
Modélisation		R	S	S
IA	S		R	S
Multijoueur	R	S		
Site Web	R		S	
Trailer		S		R

2.5 Évolution par soutenance

Dans l'ensemble la prévision qui a été faites lors de la rédaction du cahier des charges au début du projet était juste même si certains points divergents. En effet, il était très compliqué de se rendre compte, pour chaque partie du projet, combien de temps allez prendre leur implémentation. C'est pourquoi vous trouverez ci-dessous un tableau représentant l'évolution du jeu au moment des différentes soutenances.

Tâches \ Soutenance	7/03	25/04	6/06
Interface	50%	75%	100%
Logo	100%	100%	100%
Son	0%	15%	100%
Gameplay	25%	50%	100%
Game Design	50%	100%	100%
Animation	10%	75%	100%
Modélisation	25%	50%	100%
IA	25%	100%	100%
Multijoueur	25%	25%	100%
Site Web	10%	50%	100%
Trailer	0%	0%	100%

3 L'avancement du projet

3.1 Interface

3.1.1 Logo

Une des choses importantes pour le lancement d'un jeu vidéo, c'est d'avoir un logo. Un logo est un moyen de communication et d'identification indispensable à toute entreprise et à tout jeu. En effet, il constitue l'un des piliers de la communication visuelle. Et, il s'avère incontournable avec l'intensification de la concurrence sur les marchés. C'est pour ça que l'on trouvait que la conception d'un logo était une étape importante bien que ce ne soit pas le but premier du projet.

Ayant déjà des connaissances en graphisme, il nous semblait évidant que cette tache reviendra à Tom et il a donc voulu faire un logo simple et reconnaissable. Pour la conception, Tom à utilisé 2 logiciels : Photoshop et Illustrator de la suite Adobe.

FIGURE 1 – Logo du jeu

Pour la signification du logo, tout en haut on peut y apercevoir une gemme. Cette gemme est tout simplement la monnaie du jeu. Le logo entier représente un trophée qui pourrait être implémenté dans la suite du jeu, pour par exemple, débloquer une nouvelle carte de jeu et/ou des nouvelles compétences.

Pour finir, nous avons décidés de choisir des couleurs sombres pour représenter l'ambiance post-guerre du jeu. Quand au vert, le vert est la couleur « par défaut » du zombie. Si on regarde bien, la couleur des zombies présente sur la plus-part des logos est vert.

3.1.2 Menu

Avant de pouvoir jouer à un jeu, on passe d'abord par le menu principal, celui-ci sert notamment à pouvoir changer quelques paramètres comme la résolution, augmenter/diminuer le volume, changer les touches, etc...
De plus un menu permet de pouvoir implémenter une boutique si il y en a une, ajouter une rubrique sociale ou l'on peut ajouter des personnes en amis.

Tom c'est donc attelé à l'entièreté des menus présent dans Zakharov Project. Pour le menu d'accueil, afin de faire quelque chose de plus esthétique, Tom a utilisé le pack libre de droit « TextMesh Pro ». Ce pack permet notamment d'avoir un plus large panel d'option sur le texte afin de faire des boutons beaucoup plus beaux. Tom a donc commencé à implémenter 3 boutons : Jouer, Options et Quitter.

3.1.2.1 Jouer

Le bouton « Jouer » nous amène sur un sous-menu « Jouer » qui nous permet de choisir entre deux modes de jeux : Solo et Multijoueur. Le mode « Solo » nous permet de jouer tout seul tandis que le mode « Multijoueur » nous permet de rejoindre et jouer avec d'autres personnes.

3.1.2.2 Options

Un sous-menu « Options » s'affiche quand on appuis sur le bouton « Options ». Ce menu permettra aux joueurs plusieurs actions.

La première est de changer la résolution. Pour l'instant seulement trois résolution sont disponibles : 1920x1080, 1280x720 et 640x360 mais il est plutôt facile d'implémenter d'autres résolutions étant donné que le code est similaire et facilement changeable.

La deuxième action possible est d'activer ou désactiver le mode plein écran. Cette dernière action est simple à implémenter de base mais seulement si on ne changeait jamais de résolution car le code qu'on a utilisé prend en paramètre la taille de l'écran sur l'axe X et sur l'axe Y mais aussi un booléen pour savoir si cette résolution devait être en plein écran (true) ou non (false). On a donc créé des variables pour combiner le code de la résolution et du mode plein écran.
Maintenant si on désactive le mode plein écran et qu'on change la résolution, la taille de la fenêtre de jeu va changer sans changer la qualité. Mais si on active le mode plein écran et que l'on met une basse résolution le jeu va perdre en qualité mais gagner en FPS.

La troisième action effectuable par le joueur et d'augmenter et/ou diminuer la musique du menu et de gérer aussi les sons d'effets sur les boutons.

Pour finir, la dernière action présente dans se menu est de modifier les touches de déplacement du personnage, de tire et de viser.

3.1.2.3 Boutique et Profil

C'est boutons n'étant pas essentiels, on a décidé de ne pas implémenter ces fonctions pour le moment. Si nous avions le temps, le bouton Profil serait le premier implémenté.

3.1.2.4 Quitter

Ce bouton permet juste au joueur de quitter le jeu et de revenir à son bureau.

3.1.2.5 Les sons

Une fois le fond du menu créé, il fallait ajouter une musique de fond qui permet aux joueurs d'avoir une expérience encore plus immersive. Pour se faire et étant donné qu'aucun de nous avez les compétences suffisantes pour créer une musique en partant de zéro, Tom a décidé de prendre une musique libre de droits sur Youtube.

Pour en finir avec la musique du menu, Tom a retravaillé cette dernière afin d'y ajouté un bruit de feu qui crépite pour refléter le fond, des bruits de zombies et des bruits de pas. Tout ça avec un effet de son 3D qui permet aux joueurs jouant avec un casque et/ou écouteurs d'entendre les bruits à 360 degrés (entendre les bruits de pas derrière sois par exemple). Les travaux sur les sons on était effectué sur le logiciel Vegas Pro.

Ensuite, comme dis un peu avant, quand le joueur passe son pointeur de souris ou quand il clique sur un bouton, un petit effet se produit. On a rajouté un son, un pour les cliques et un autre son pour le passage du pointeur uniquement, qui s'applique et rajoute un plus au jeu. Les sons ont aussi été faits sur Vegas Pro.

3.1.3 Vidéo d'arrière plan

La vidéo utilisée en fond sur le menu a été faite par Lucas en utilisant différents logiciels. Les modèles 3D utilisés viennent de la même bibliothèque Quixel que pour le reste des assets du jeu mais ont été importés pour les besoins de la scène en qualité 8K, une qualité impossible à utiliser dans le jeu. De plus, le rendu à été effectué sur Unreal Engine, principal rival de Unity en termes d'édition de jeux mais proposant bien plus d'options sur le plan de la production cinéma en 3D.

C'est donc grâce à ce logiciel que Lucas a rendu la séquence du menu. Pour faire un rendu, on met tous les éléments en place dans une scène comme on le ferait pour créer un niveau sur Unity, puis on ajuste les différents effets comme l'éclairage, la couleur, le flou d'arrière plan, les particules flottantes pour obtenir l'ambiance recherchée. Une fois tous les paramètres ajustés on lance un rendu dans le format jpg ou png avec une fréquence d'image par seconde définie.

A la fin du processus on obtient un dossier contenant pour 20 secondes de séquence un peu plus de 1200 images en 4K. Grâce à un logiciel tiers, on fusionne les images en une vidéo compressée. La vidéo a ensuite été exportée vers Adobe Premiere Pro pour retoucher les couleurs, la luminosité et la netteté. Le logiciel de montage a également permis de mettre la vidéo en boucle afin que le joueur ne puisse pas voir de coupure lorsque la séquence reprend du début. La vidéo a ensuite été transférée à Tom qui l'a implémentée dans le menu.

3.2 L'environnement

3.2.1 Level Design

Ayant expérimenté différents outils de level design dont celui de unity par le passé, Lucas décidé de commencer pour sa part le projet avec le design de la map. Un des outils que qu'il a utilisés par le passé était la bibliothèque Megascans proposée par Quixel. L'entreprise propose une bibliothèque de modèles et textures 3D de haute qualité. Pour obtenir ce rendu, ils voyagent dans le monde entier et photoscannent tous les environnements et proposent ainsi une qualité photoréaliste. La bibliothèque est utilisée par de nombreux studios de jeux vidéo comme de cinéma, notamment Rockstar Games avec Red Dead Redemption II ou encore les studios Walt Disney pour l'adaptation live action du livre de la jungle. (9)

Ayant réfléchi préalablement au style du jeu, nous nous sommes mis d'accord sur un jeu aux graphismes plutôt réalistes et en 3D. Nous étions conscients que cela impliquait bien plus de travail qu'un jeu en 2D ou même qu'un jeu en 3D stylisé et aux graphismes simples mais cela nous paraissait être la meilleure solution pour pouvoir mettre en place l'ambiance que nous voulions.

Cet outil professionnel nous a donc permis d'avoir un rendu assez réaliste et qui collait à l'idée que nous avions en tête au début du projet. Le problème avec une telle qualité est qu'il faut que tout concorde. Ainsi dès que l'on avait besoin d'un modèle 3D n'étant pas disponible dans la bibliothèque Quixel, il fallait rechercher celui-ci et en trouver d'aussi réalistes que le reste du jeu pour ne pas se retrouver avec des différences stylistiques au sein du projet.

Afin de tirer le meilleur parti des différents modèles et textures, nous utilisons pour ce projet le HDRP inclus dans Unity. La HD Render Pipeline est une méthode de rendu basée sur la physique permettant des éclairages réalistes calculés de la même manière qu'ils fonctionnent dans le monde réel et globalement une qualité plus élevée en ce qui concerne les textures et les effets de particules. Le HDRP nous permet également l'utilisation de l'outil de création de terrain qui sera évoqué plus tard. Le choix de ce mode de rendu était un assez gros risque puisque les projets créés en HDRP ne sont pas compatibles avec la Render Pipeline de base. Une fois commencé le projet ne pouvait donc plus être changé.

Quant à la map dans son ensemble, voici en annexe le plan initial de la map (10). Le plan fut établi dès le début du projet et n'a finalement pas changé, la map s'est construite autour de celui-ci pendant tout le projet. A la manière des jeux Call of Duty zombie, la map contient différents environnements, tous placés dans un thème post-apocalyptique. Le level doit être formé de sorte à ce que le joueur puisse tourner en rond lorsqu'il est poursuivi par la horde de zombies. Notre level se décompose ainsi : une partie intérieure avec la prison et l'entrepôt ainsi qu'une partie extérieure qui est la cour de la prison.

3.2.1.1 Intérieur

Lucas a dans un premier temps développé la partie prison. Cette première portion a pendant longtemps servi de terrain de test pour les différentes fonctionnalités du jeu. Cette partie à été réfléchie de sorte à ce que les assets qu'elle contient puissent être assez universels et réutilisés dans d'autres parties. En effet, les assets 3D et textures associées ont pu servir également dans la partie entrepôt ce qui nous a évité d'importer plus d'assets (ce

qui aurait augmenté le poids du jeu). Il était donc important de prendre en considération tout ces paramètres avant de commencer le design de la prison.

Par la suite Lucas a complété la partie entrepôt, celle-ci contenant un grand nombre d'objets 3D, elle s'est avérée la plus problématique au niveau performances. Un problème qui fut résolu par la suite grâce aux techniques évoquées plus loin dans ce rapport. La partie entrepôt utilise dans sa globalité des modèles 3D repris de la partie prison comme expliqué précédemment. Ainsi, nous n'avons pas eu de problèmes de stockage lors de l'intégration de cette partie puisque le nombre de modèles 3D et textures n'a pas augmenté.

Lucas a également ajouté des lampes néon dispersées au plafond sur tout le niveau. Ces lampes ont nécessité l'utilisation des shaders de unity, un outil de coding visuel permettant de créer des textures en reliant différentes cases de propriétés. Il a donc réussi à créer un matériau ressemblant à un néon avec l'effet de bloom qui permet une simulation de lumière aveuglante sur la caméra du joueur.

Les parties intérieures de la map étaient en résumé plutôt faciles à mettre en place et requéraient l'usage des outils basiques de unity pour placer les assets. Le plus compliqué était de paramétrer les textures et leur albedo, roughness, et metallic afin d'obtenir l'ambiance recherchée. Ces trois paramètres liés aux textures des objets 3D permettent de changer totalement le rendu de celui-ci pour être plus coloré, plus terne, plus sombre ou plus lumineux.

3.2.1.2 Extérieur

L'extérieur en revanche a nécessité l'utilisation d'outils plus compliqués comme le créateur de terrain qui a permis sur la fin du projet de finaliser la map avec des reliefs et de la végétation sur la partie extérieure. Cette partie a donc nécessité l'importation d'autres modèles 3D et ne réutilise évidemment pas la plupart des assets de la partie intérieure.

C'est donc lors de l'intégration de cette partie que nous avons commencé à avoir des problèmes liés au partage via Git. Le stockage du git était plein et les commits excédaient la taille maximale. Nous avons donc opté pour un git professionnel offrant plus de stockage, le projet pesant à ce moment-là plus de 20 Go. Cette solution à donc résolu nos problèmes. Nous avions également quelques problèmes de performance liés aux effets de vent dans la végétation qui faisaient bouger les feuilles. Cet effet mineur à donc été désactivé pour plus de performances.

Enfin quelques ajustements ont été nécessaires une fois le niveau fini pour des question d'équilibrage de la difficulté face aux zombies (obstacles présents sur le niveau, endroits d'apparition des zombies par rapport au joueur, ...). Il ne faut pas que les zombies puissent par exemple apparaitre sur un point d'apparition alors que le joueur se trouve tout près. Toutes ces choses ont nécessité de nombreux tests pour avoir un jeu à la difficulté jouable, ni trop élevée, ni trop basse.

Une fois cette dernière partie rajoutée, la map était complète. On arrivait à l'ambiance que l'on voulait recréer dans notre jeu : une ambiance assez sombre par le biais de l'environnement au look post-apocalyptique et différents effets d'éclairages. L'éclairage joue en effet un rôle important avec une combinaison de post-

processing, brouillard volumétrique et ombres projetées.

Le brouillard volumétrique permet des raies de lumières qui ajoutent à l'ambiance tandis que les ombres projetées font contraste. Chaque mesh sur laquelle arrive une lumière produira une ombre réaliste. On combine le tout à une skybox personnalisée afin d'avoir une lune qui passe à travers le brouillard. La skybox est une image jpeg contenant des nuages et un soleil ou une lune qui va être enveloppée autour du niveau afin de donner l'illusion d'un ciel. Le tout permet d'obtenir l'ambiance escomptée.

3.2.2 Optimisation

Une fois la map complétée, nous l'avons transféré sur le git afin de pouvoir la partager entre les différents membres du groupe et pour que tout le monde travaille sur la même version du projet. Lors de l'import de la map sur des ordinateurs divers, nous nous sommes rendu compte que celle-ci demandait beaucoup de ressources à la machine notamment concernant la carte graphique, ce qui affectait la fluidité du jeu et le rendait moins fluide avec une perte d'images par seconde sur certains ordinateurs. En supprimant puis rétablissant certains éléments du niveau, nous avons déterminé que les baisses de performances venaient principalement des effets de lumières et de la qualité des modèles 3D du niveau. Nous avons fait de nombreuses recherches sur les différentes techniques d'optimisation d'un jeu Unity et avons mis en place les plus efficaces d'entre elles.

Pour remédier aux performances médiocre, Lucas a donc mis en place une optimisation LOD. Chaque objet 3D présent dans une scène est composé de triangles qui définissent ses contours. Lors de son importation, on peut choisir si le modèle 3D est importé avec ou sans LOD variables. Si l'on choisit d'importer un

objet avec plusieurs niveaux de LOD, alors cela prendra plus de stockage.

En revanche, ces différents niveaux de détails permettent par la suite l'implémentation de l'optimisation LOD, elle permet de réduire le nombre de triangles composant un modèle 3D lorsqu'on s'en éloigne (**??**). Au fur et à mesure que le joueur s'éloigne de l'asset, celui-ci devient de moins en moins détaillé, son nombre de triangles et donc son niveau de détail réduit. En effet si le joueur est loin, on n'a plus besoin d'avoir quelque chose d'extrêmement détaillé. Tout ces changements sont ajustables avec l'outil de changement de LOD intégré à Unity, une barre permet de régler à partir de quelle distance les différentes version d'un modèle 3D s'affichent.

Une fois le joueur à bonne distance du modèle, nous n'avons plus qu'une version simplifiée du modèle 3D de base, on réduit donc le nombre de triangles le composant (la qualité du LOD) ce qui, une fois appliqué à tous les éléments du niveau, permet de meilleures performances en jeu. Cet ajout a permis une augmentation considérable des performances du jeu mais n'était pas suffisant.

Lucas a donc implémenté une autre technique qui peut se résumer ainsi : si un modèle 3D et sa texture ne se trouvent pas dans le champ de vision du joueur, alors ces derniers sont déchargés et stockés dans la mémoire vive afin de pouvoir être rechargés rapidement. Si un modèle 3D éclipse complètement un autre modèle 3D derrière lui, alors l'objet obstrué est déchargé. Les triangles évoqués précédemment sont donc inexistants en dehors du champ de vision de l'utilisateur ce qui permet de meilleures performances.

C'est ce qu'on appelle l'occlusion culling, une option intégrée de Unity. On peut « Bake » un niveau afin de générer le data lié à l'occlusion culling. Si on change le moindre modèle 3D sur le niveau, il faut recommencer le process de Bake. Cette dernière technique s'est révélée être la plus efficace pour soulager la carte graphique et améliorer le nombre d'images par seconde.

Toutes ces techniques sont particulièrement utiles maintenant que la taille du niveau a considérablement augmenté avec un nombre d'objets 3D et de prefabs triplé par rapport aux premières itérations de la map. L'objectif était une fois arrivé à l'état final d'avoir un jeu fluide peu importe le nombre de zombies présents. Comme dit plus haut ces techniques ont un désavantage qui est le stockage pris par les différentes données, les différents LOD ainsi que le bake de l'occlusion culling prennent de la place mais permettent d'augmenter les performances. Ce désavantage s'applique à toutes les optimisations, par exemple si l'on avait choisi d'avoir un éclairage non dynamique il aurait fallu bake l'éclairage, les données du bake auraient pris plus de place. Nous avons fait le choix en revanche de ne pas utiliser cette dernière optimisation puisqu'elle enlève à l'immersion du jeu, comparé aux autres optimisations qui sont mineures et à peine visibles.

3.3 Gameplay

3.3.1 Insertion du joueur

3.3.1.1 Fonctionnement

Pour commencer, il a fallu créer un joueur qui nous sert de base pour notre code, dans le cas de Tom pour les déplacements. Tom a donc importé, grâce au site Mixamo, qui est un site d'animation et de personnage 3D en libre-service, un personnage 3D.

Avant de passer sur la programmation des déplacements, Tom voulait que la « hitbox », la zone sensible du personnage, soit dissociée de la partie visuelle du joueur. Cette étape n'est pas nécessaire mais elle permet que la partie visuelle n'influe pas sur la partie physique du joueur et inversement. C'est pourquoi Tom a dû supprimer le « Mesh Renderer » et le « Mesh Filter » du personnage 3D afin d'enlever le visuel et de garder uniquement le « Collider », c'est-à-dire la « hitbox ». Une fois cela fait, Tom a dupliqué le joueur à l'intérieur du joueur pour qu'il soit parent de l'autre, mais cette fois ci en enlevant le « Collider » et en rajoutant le « Mesh Renderer » et le « Mesh Filter ». Grace à cette étape on a donc obtenu un joueur dont le visuel et la « hitbox » étaient dissociés.

Pour finir, Tom a aussi rajouté un « Rigidbody » au joueur sur lequel il a gelé les rotations en X, Y et Z car les rotations se feront grâce à la caméra. Le « Rigidbody » sert aussi à toute la partie physique du joueur (pour qu'il soit soumis à la gravité par exemple).

3.3.1.2 Apparence

L'univers et l'ambiance du jeu est importante pour nous. En effet, comme dit précédemment notre jeu se déroule lors de la Seconde Guerre mondial côté soviétique. Corentin a donc choisi un habitant soviétique de l'époque comme model de notre joueur.

De plus, Corentin a ajouté une barre de vie pour chaque joueur, ainsi maintenant les zombies font des dégâts aux joueurs lors de leurs attaques. Si le joueur arrive à zéro de point de vie, il meurt et définitivement.

Au début, lors de parties en multijoueur, nous voulions que les joueurs puissent se réanimer entre eux. Cependant, par manque de temps, cette idée a été abandonnée. Quand le joueur meurt il arrive sur un écran de fin qui lui affiche le score final ainsi que la possibilité de relancer une partie ou de revenir au menu.

3.3.1.3 Animation

Pour la dernière soutenance, Flavien à revu complètement la gestion des déplacements avec les animations. Il a créé un nouvel Animator à 2 Dimensions (X et Z) ce qui a permis de gérer plus facilement et plus proprement les animations gauche, droites, en arrière, et en avant, en fonctions des touches. (ANNEXE 1)

Le joueur peut également courir en avançant et en appuyant sur Lshift. L'Animator fait par Flavien permet également d'avoir une fluidité dans la marche et la course du joueur.

Un autre gros problème que nous avions lors de la deuxième soutenance, est la tenu de l'arme dans les mains du joueur. Notre animation prise sur Mixamo de la tenu de l'arme dans les mains

ne collaient pas avec notre modèle (les mains du modèle était retournées). Flavien à donc recréer une animation simple de la tenue d'une arme sur Blender (Figure2), puis l'a importer sur Unity.

De plus il a ajouté un Layer et un masque à l'Animator du joueur. Le masque (Figure 2) combiner à un Layer permet de jouer dans notre cas que l'animation du haut du corps. En le combinant avec un deuxième Layer (ici, nos déplacement), deux animations sont jouer en même temps.

Dans notre cas, ce système nous permet d'avoir le mouvement des jambes seulement sur le bas du corps, et la tenue de l'arme seulement sur le haut du corps. Cela rend l'animation du joueur plus propre et plus réaliste.

Flavien à également gérer le changement de la position des mains en fonction de l'arme du joueur. Avec une animation faite sur Blender, si le joueur tiens le pistolet, la position de ses mains change.

3.3.2 Déplacement et caméra

En partant sur l'idée d'un jeu de type FPS (First Person Shooter), il fallait donc commencer par implémenter tous les déplacements essentiels tel que la marche avant et arrière, la course, le saut et l'accroupissement du joueur. Après toute la partie d'initialisation du joueur, Tom a donc pu passer à la programmation des déplacements.

Pour commencer, afin de faire un code propre et plus lisible, Tom a décidé de créer deux sripts : PlayerController et PlayerMotor. PlayerController qui est le script qui détecte les entrées. C'est-à-dire les touches du clavier et de la souris sur lesquelles

on va appuyer. PlayerMotor va quant à lui être responsable de la physique du personnage, les déplacements ou d'autres actions de ce genre. C'est deux scripts seront donc complémentaire car les déplacements seront effectués en lisant les entrées. Vu que ces deux scripts ne peuvent marchés l'un sans l'autre, Tom a dû utiliser un « RequireComponent » qui permet d'indiquer à un script qu'il ne peut pas fonctionner sans un autre.

Les déplacements n'ont pas été un problème car tout se faisait grâce à la méthode « Translate » qui me permet de gérer les déplacements en X, Y et Z. Il fallait juste changer mes variables en fonction de si le personnage courait, reculait ou marchait. Le seul problème qui est survenu était pour le saut. Il fallait que le code reconnaisse que le joueur n'était pas en l'air pour sauter afin qu'il ne puisse pas sauter à l'infini.

Pour se faire Tom avais comme idée d'utiliser les « Raycast » qui sont, pour résumer en deux mots, des lignes imaginaires que l'on créer. Tom voulait donc créer un raycast allant du milieu du joueur vers le bas et observer en continue si cette ligne touche le sol ou non pour déterminer si le joueur touche le sol. Cette méthode aurait pu marcher si le sol sous le joueur avait été toujours plat ou très légèrement incliné.

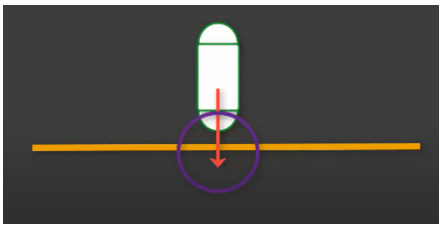

FIGURE 2 – Le raycast touche le sol

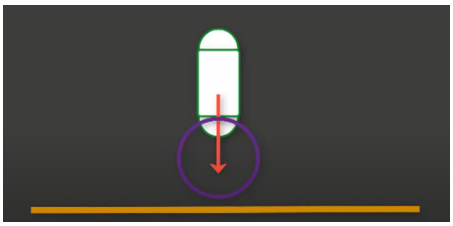

FIGURE 3 – Le raycast ne touche pas le sol

Or si le sol est très incliné, un coté du joueur pourrait toucher le sol mais pas forcément le raycast. De plus, en fonction de la

taille du raycast utilisé, le joueur pourrait monter sur des pentes qui sont impossible à monter dans la vraie vie.

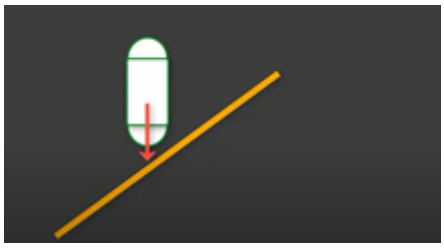

FIGURE 4 – Sol pentu

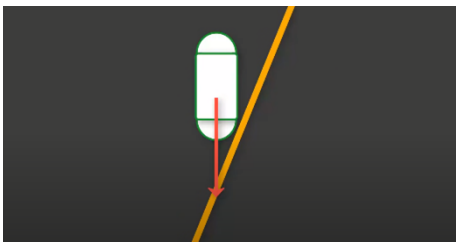

FIGURE 5 – Sol très pentu

Pour remédier à se problème, la ou le raycast effectué une seule vérification, j'ai décidé dans effectuer plusieurs : si on détecte un sol au centre du personnage, centre de l'extrémité X et sur le centre de l'extrémité Z.

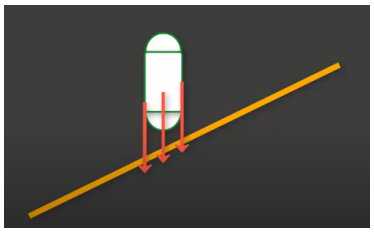

FIGURE 6 – Triple vérification

Un peu plus tard, Tom a voulu faire un changement total du système de mouvement et revoir ce dernier entièrement pour faire des mouvements plus réalistes. Pour ce nouveau système, on pouvait toujours bien évidemment se déplacer en avant, en arrière, sur les coter, courir, sauter et s'accroupir mais contrairement à l'ancien système, la caméra bougeait afin de créer un effet plus réaliste qui s'approche de la réalité. Quand on courait et/ou quand on marchait, on avait l'impression que la tête du personnage bougeait (un plus gros effet quand on court).

Malheureusement, Tom à rencontrer un problème lors de la mise en groupe du projet. C'est-à-dire que, sur sa branche, les nouveaux mouvements marchaient parfaitement, mais lors de la mise en pratique sur les autres branches du projets les déplacements ne marchait plus. Cela venait du fait que la majorité de nos codes se base sur les deux scripts de mouvement du tout début. On devait donc changer une grande partie de nos codes pour régler le problème et c'est pour ça que Tom a abandonné l'idée de mouvement réaliste.

Ensuite, une fois les déplacements fait, Tom voulais implémenter une glissade mais il s'est dit qu'elle n'était pas essentielle au commencement du jeu. C'est pourquoi il a préféré passer sur la gestion de la caméra afin que ce dernier bouge en fonction de notre souris pour orienter la vue du joueur. Une fois tout cela effectué, Tom a essayé d'implémenter quelques animations pour la course, la marche, ect... encore grâce au site Mixamo. Tom est vite passé sur cette étape qui ne lui semblait pas très importante pour le début du projet, il l'a donc laissé en état de brouillon pour finir les animations plus tard.

3.3.3 Les Spawners

Une fois les premiers zombies créer, il a fallut se pencher sur leur l'apparition dans l'environnement. Pour notre jeu il fallait trouver un moyen efficace de faire apparaître les zombies au cours d'une vague.

Dans de nombreux jeux l'apparition des zombies se fait loin du regard du joueur. À aucun moment le joueur ne voit apparaître le zombie. Nous nous sommes donc pencher sur les différentes possibilités que nous avions.

Un moyen simple de faire apparaître les zombies sans que le joueur le voit, est de créer des points d'apparition, des spawner, dans des pièces sombres inaccessibles aux joueurs mais d'où les zombies peuvent sortir.

C'est Flavien qui s'est pencher sur cette tache en créant un objet Spawner auquel est attaché un script et les zombies. Dans le script, il y a une classe Spawner qui, grâce à un compteur, instancie un nombre donné de zombie à la position du Spawner, séparé par un intervalle de temps. Une fois le zombie instancier il avance en direction du joueur le plus proche.

Pour la deuxième soutenance, après avoir ajouté de nouveaux zombies, et donc pour permettre la diversité de zombies, le script des spawner à été légèrement modifié. En effet, les spawner font maintenant apparaître un des 8 types d'ennemis de façon aléatoire.

3.3.4 ZCoins

Comme évoqué dans les autres parties la difficulté va augmenter au fil du temps, au fur et à mesure des vagues les zombies vont apparaître plus fréquemment et en plus grand nombre. Pour surmonter ce challenge, le joueur doit avoir la possibilité d'améliorer son arsenal. C'est pourquoi Lucas a implémenté un système d'argent.

Lorsque le zombie meurt, il laisse au sol un icône 3D qui représente l'argent. C'est un petit logo du jeu, dessiner par Tom, puis modéliser et animer sur Blender par Flavien. Lorsqu'on marche dessus, le joueur gagne de l'argent.

Cet argent est très important pour le joueur, sa survie en dépend puisqu'il lui permet d'acheter des armes plus puissantes, de se ravitailler en munitions et en soins s'il perd de la vie et ainsi de continuer plus longtemps et obtenir un meilleur score.

Les packs de soins et munitions seront disponibles à l'achat dans certains endroits du niveau. Pour que le joueur puisse se repérer plus facilement, nous avons ajouté des néons aux endroits de vente.

3.3.5 Le jeu

Pour ce qui est du gameplay de manière générale, il est plutot simple. Le jeu commence et tant que tous les joueurs ne sont pas morts, des zombies et des boss apparaissent dans la map par vague. Plus le joueur avance dans les vagues, plus les ennemies sont nombreuses et redoutables.

Il a donc fallu faire un script permettant de gérer le jeu. C'est Flavien qui a implémenté ce script. Une fois les spawners trouver, une fonction permet de lancer une vague en faisant apparaître un nombre donné de zombie par spawner.

Ensuite, après le lancement de la première vague, une autre fonction permet de savoir si tous les zombies présents sur la map sont morts. Si c'est le cas, le nombre de zombie par spawner s'incrémente et une nouvelle vague commence.

Pour la deuxième soutenance, Flavien à travailler à nouveaux sur le script du jeu et des spawner et à régler certains problèmes.

Par exemple, un problème à régler était que lorsqu'une vague commençait et que les joueurs tuaient tous les zombies sur la map, avant même que les spawn de zombies ne soient tous terminé, une nouvelle vague commençait. Comme la vague précédente n'était pas terminer, deux vagues se jouaient alors simultanément. Flavien à donc fixer se problème.

Flavien et Corentin ont également mis en place la partie score des joueurs. Lorsqu'un joueur élimine un zombie il gagne 100 points.

Finalement, plus les joueurs tuent de zombies, plus ils avancent dans les vagues, et plus les zombies sont nombreux.

3.4 Ennemies

3.4.1 Intelligence artificielle

L'IA est à la base du projet car bien évidemment nos zombies ne sont pas contrôlés par des humains ! Il a donc fallu implémenter un système d'intelligence artificielle afin de créer la base du gameplay du jeu. C'est principalement Flavien qui à travailler sur cette tache. Ayant déjà joué à quelques jeux de zombies, Flavien avais déjà un bonne exemple en tête pour créer ces monstres.

Avant toute chose, il fallut trouver les zombies (un personnage à animé). Ayant ni les connaissances en modélisation, ni le temps, nous nous sommes tournés vers une solution plus efficace et qui nous a permis de coder plus rapidement : le pré-set libre de droit.

Nous avons donc utilisé Mixamo (LIEN), une banque de personnage et d'animation libre de droit développé pour Unity. Grace cette banque nous avons eu accès à de nombreux models pour nos zombies ainsi que beaucoup d'animations.

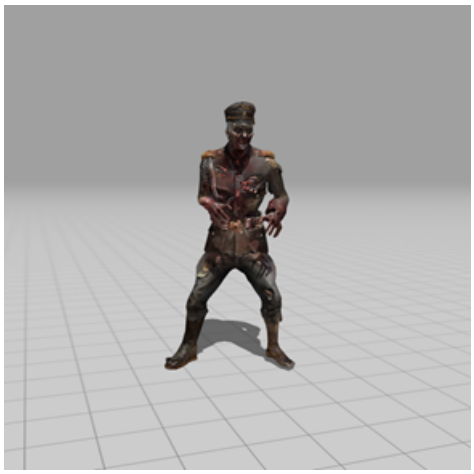

FIGURE 7 – Model d'un zombie

Une fois que l'on a eu notre premier prefab de zombie, avec notre corps, Flavien a pu commencer à coder et à mettre en place le système d'intelligence artificielle. Heureusement pour nous, le zombie ne réfléchis pas trop. Il chercher seulement à rejoindre et tuer le joueur le plus proche.

Après plusieurs recherche Flavien à également trouver qu'il était plus simple d'implémenter mouvement dans l'espace et animation en même temps.

Le système de déplacement se fait donc grâce un script C, un Character Controller et un Animator c'est à dire un outils qui permet d'animer l'objet sur lequel il est attaché en fonction de paramètres.

Pour ce qui est de l'Animator, Flavien à trouver des animations de zombies sur Mixamo.

Dans le script, via un système de Tag, le zombie cherche toujours à trouver le joueur le plus proche de lui. Et une fois qu'il l'a trouvé, il se dirige vers lui. Grâce aux paramètres dans l'Animator et à des conditions dans le script, le zombie est capable, en évaluant la distance entre lui et sa cible, de faire telle ou telle animation.

Si cette distance est supérieur à 10m, il va marcher. Si elle devient inférieur, il va faire un cri puis se mettre à courir vers le joueur. Enfin si la distance joueur-zombie est inférieure à 1m, le zombie lance une attaque sur le joueur. Les transitions entre les animations ont été fluidifié pour avoir un mouvement plus propre.

En plus de ça, une vie au zombie ainsi que deux fonctions : une qui permet de savoir si le zombie est mort, et une qui lui fait perdre de la vie, ont été implémenter.

Lors la première soutenance, nous avions donc un premier zombie qui pouvait se déplacer et qui était animé. Mais nous avions un problème, notre intelligence artificielle ne pouvait que se diriger en ligne droite du point d'apparition jusqu'au joueur. Si un objet se trouvait entre le zombie et le joueur, le zombie restait bloqué.

Pour régler se problème, Flavien à donc du mettre en place un Mesh de navigation sur la map, c'est à dire l'environnement dans lequel les zombies peuvent se déplacer. Ensuite il a fallu adapter le script de déplacement des zombies. Mettre en place le NavMesh restait plutôt simple une fois la documentation lu, mais adapter le script à poser de nouveaux problèmes, notamment dans la gestion des animations en fonction de la distance.

Flavien à donc mis en place le NavMesh sur la map (zone bleue sur la Figure 1), puis adapter les scripts à l'aide d'un NavMesh Agent, un composant qui permet à un objet de trouver le chemin le plus court entre la position de l'objet en question et une destination, et de s'y déplacer. Le NavMesh Agent est très puissant puisque si la destination bouge, ce qui notre cas, l'objet (ici le zombie) peut ajuster le chemin a prendre. Dans tous les cas, les zombies se déplaceront jusqu'au joueur le plus proche en prenant le chemin le plus court, sans rester bloqué contre un mur ou encore un objet.

Pour la première soutenance, nous n'avions qu'un seul ennemi, un seul zombie. Pour la deuxième soutenance et pour la dernière soutenance, un gros travail à été fait par Flavien, pour diversifier les ennemies.

Nous avons maintenant 2 modèles de zombies pour 4 types de zombies différents : En effet nous avons un modèle de zombie capitaine ainsi qu'un modèle de zombie prisonnier. Avec ces 2 modèles, Flavien à créer 6 types de zombies, grâce à 6 Animator différents. Nous avons donc 6 types de zombies, animé de façon unique, tous fonctionnels dans le NavMesh.

Les scripts de nos zombies étant très similaire, le même script à été utilisé pour tous les zombies. Ce qui réduis l'utilisation de la mémoire du projet.

Pour ajouter encore plus de diversité à nos ennemies, Flavien à créer 2 autres ennemies, 2 ennemies plus puissant, avec 2 modèles différent et 2 autres Animator unique. Ces ennemies sont eux aussi fonctionnels avec le NavMesh. Au total, notre jeu contient 8 ennemies différents, tous animé et tous fonctionnels avec le mesh de navigation.

Pour la dernière soutenance Flavien à finis l'IA et les zombies en modifiant le système d'animation de certains zombies. Auparavant, lorsque les zombies atteignaient le joueur, ils attaquaient. Maintenant, ils peuvent lancer un coup de points tout en marchant.

Cette légère modification qui rend les animation plus propre et le mouvement plus cohérent à été fait grâce aux masques et aux layers des Animators qui permettent de jouer plusieurs animations en simultané.

Pour finir sur l'IA et les zombies, toute la partis sons, cris, grognement à été ajouté pour la dernière soutenance.

Notre jeu est donc doter d'un système d'intelligence artificielle complet : animation, sons et déplacement ; fluide et cohérent pour le joueur.

3.4.2 Ragdoll

Dans la continuité du travail réalisé sur les zombies, Lucas a implémenté différentes fonctionnalités qui suivent la mort du zombie lorsqu'il est tué par le joueur.

La première est le système de ragdoll (corps simulé physiquement). Lorsque le joueur tue le zombie, le programme stoppe toute fonctionnalité liée au fonctionnement du zombie. Ce dernier n'est donc plus animé et l'IA qui le régit est supprimée. Il ne nous reste donc plus qu'un corps auquel nous appliquons un ragdoll. Cela permet lors de la mort du zombie de simuler physiquement son modèle 3D qui est tenu par différents joints (articulations) aux endroits où elle seraient placées sur un corps normal. Ces joints évitent que des membres se plient de façon non réaliste.

Mais alors à quoi sert le ragdoll ? Le système évite d'avoir à animer la mort du zombie, en effet l'animation pourrait faire passer le modèle 3D à travers des murs selon l'endroit où il est tué et d'autres bugs apparaîtraient. Par ailleurs, la solution de faire disparaître le zombie dès sa mort aurait été plus simple mais enlève à l'immersion du jeu. Aussi, simuler physiquement permet d'avoir moins de répétitivité dans le jeu en ajoutant un élément qui est aléatoire (11).

Pour mettre en place l'effet, on utilise l'asset physique ragdoll, fourni par Unity. L'asset permet d'associer chaque joint du rig du personnage à une case qui s'occupe de créer les hitbox de chaque membre et de leur attribuer une masse réaliste.

Quelques problèmes nous sont apparus lors de cette mise en place puisque les hitbox générées ne sont pas toujours fidèles au modèle et nécessitent des ajustements. L'outil permet par ailleurs de régler l'angle auquel peuvent se plier les différents membres et sur quel angle x, y, z, ils peuvent tourner afin d'obtenir des mouvements plus réalistes. Le ragdoll s'active donc dès que la vie du zombie est inférieure à 0. Pour des questions de performances le corps disparait au bout de 15 secondes et un coin dont aura besoin le joueur sera lâchés au sol quand un zombie meurt.

3.4.3 Sons

Au niveau des sons faits par les zombies, c'est Flavien qui à travailler dessus pour la troisième soutenance. Chacun des zombies et des boss font des cris, des grognement différents.

Flavien à également mis des sons de bruits de pas lors des déplacements du joueur.

3.5 Armes

3.5.1 Système de tir

Le système de tir, implémenté par Corentin, est l'essence même du jeu, il permet aux armes de savoir si un zombie est touché. Il est possible grâce à un raycast. Celui-ci part de la caméra du joueur vers l'avant à une distance qui dépend de la variable « weapon.range », qui est rien d'autre que la distance que peut parcourir la balle de l'arme actuelle. Chaque arme possède une distance de tir maximal mais on y reviendra un peu plus tard.

Autre partie très importante, les layers. Ce sont des masques qui forment « une famille » pour tous les « GameObject » qui le portent. Ils permettent de choisir avec quelle famille d'objet le raycast doit interagir. En effet on veut éviter de pouvoir tirer sur ses coéquipiers. Tout d'abord, le script « Player Shoot » vérifie si le joueur presse le bouton « Mouse 1 » qui correspond au bouton gauche de la souris. Si le bouton est pressé alors la méthode « Shoot » est appelé. Evidemment, cette méthode est appelée jusqu'à que le joueur lâche le bouton gauche de la souris. Elle permet de tirer un raycast et vérifier si l'objet touché est un zombie, ceci grâce à son tag.

Point important, la méthode est appelée uniquement du côté client grâce à « [Client] » devant la définition de la méthode. Au contraire, le résultat de cette méthode est « CmdPlayerShot » qui comme son nom l'indique est une commande qui est appelée par le serveur grâce à « [Command] ». Ainsi lorsque le raycast touche un zombie, un message est envoyé au serveur, celui-ci perd de la vie et le serveur préviens les autres clients connectés.

3.5.2 Implémentation

L'affichage du pistolet se fait grâce à une deuxième caméra qui est l'enfant de l'objet « Weapon Holder ». Cet objet va permettre plus tard au joueur de porter des armes de différentes taille et modèle. La caméra de l'arme affiche seulement l'arme par-dessus la caméra du joueur.

De plus, le pistolet possède aussi des particules, un flash et de la fumée au bout sortent du canon lorsque le joueur tir. Pour cela, un petit ajout de script « Weapon Graphics » qui permet de stocker les deux particules et une modification de « Weapon Manager » pour obtenir la partie graphique qui vient d'être créé. De plus, si la balle touche une autre surface qu'un zombie, des étincelles sont éjectées de l'objet touché. Tous ces effets rajoutent un certain réalisme au jeu et une meilleure expérience. De plus, des sons sont joués lors du tir, du rechargement et quand le chargeur est vide.

3.5.3 Models

Les armes sont une partie essentielle dans le gameplay du jeu. C'est pour cela que nous avons implémenté différentes armes. Pour plus de cohérences avec le thème de notre jeu, nous avons choisi trois armes : Le Makarov (13), c'est un pistolet créé 1948 est largement utilisé par les soviétiques, le PPSh-41(14) , c'est un pistolet-mitrailleur soviétique utilisé pendant la Seconde Guerre mondiale ainsi que la SKS (15), c'est une carabine semi-automatique soviétique conçue en 1945. A l'avenir de nouvelles sont susceptibles d'être ajoutées.

Nous avions l'intention de faire en sorte que le joueur tienne l'arme dans les mains pour plus de réalisme. Cependant, après de très longues heures à essayer de bien la positionner et de cherche sur internet, on a compris qu'il était très compliqué de le faire proprement. De plus, après l'ajout des animations du joueur, l'arme se déplaçait aussi de façon beaucoup trop aléatoire. C'est pourquoi on a décidé de laisser l'arme face au joueur en vu FPS.

3.5.4 Statistique armes

Pour cela nous avons donc utilisé des « scriptable object » qui permettent de créer différentes statistiques pour chaque arme (dégâts, portée, taille du chargeur, ...). De plus, on a ajouté un système de chargeur aux armes. On a donc un nombre de balle limité et lorsque que le chargeur est vide on est dans l'incapacité de tirer jusqu'au rechargement de celle-ci en appuyant sur la touche R.

3.6 Multijoueur

Le multijoueur est une fonctionnalité particulièrement complexe à implémenter et peut vite entrainer de nombreux bugs. C'est pour cela que, dès le début du projet Corentin a décidé de s'intéresser de plus près à comment l'implémenter. Après de nombreuses recherches il s'est rendu compte que le multijoueur n'est pas qu'une simple fonctionnalité à ajouter à la fin du développement, mais bien de nombreuses modifications afin que chaque partie du jeu fonctionnent en multijoueur. À la suite de cette découverte, Corentin a compris que le multijoueur est enfaite la base du jeu et qu'il était important de le développer dès le début.

3.6.1 l'API

Pour commencer avec le multijoueur il a fallu choisir une solution de multijoueur. Il existe différentes solutions de multijoueur fonctionnant sur Unity, trois choix s'est offert à nous : UNet, Photon et Mirror. UNet est la solution la plus connue car elle était le système officiel de Unity. Cependant, en 2018 UNet a été arrêté, le rendant impossible à utiliser. Une solution alternative ressemblant fortement à UNet existe, elle s'appelle Mirror. C'est une API open source, qui permet de mettre en place un système de multijoueur facilement sur Unity. Il permet de faire communiquer toutes les instances entre elles. L'utilisation de Mirror paraissait plus simple, Corentin l'a donc privilégié par rapport à Photon. Sa mise en place sur Unity a été plutôt simple car directement disponible sur l'asset store.

3.6.2 Implémentation

Ensuite, lorsque la décision de l'API utilisée et sa mise en place sur Unity a été faite, Corentin a pu commencer la mise en place d'un début de multijoueur. La base du fonctionnement de Mirror ce fait avec un objet « Network Manager » à placer sur la scène. Cet objet doit contenir un composant du même nom qui permet de gérer l'apparition des joueurs, la connexion au serveur, la synchronisation entre les différentes instances et bien d'autre encore. C'est en quelque sorte le cerveau du multijoueur ».

De plus, un autre composant nécessaire est « Network Manager HUD » qui permet d'afficher le HUD de connexion de Mirror. Corentin a utilisé ce HUD seulement lors du développement. En effet, plus tard, il a été remplacer par notre propre menu de jeu plus agréable pour le joueur.

Enfin, après tout cela implémentés une esquisse de multijoueur était née. On pouvait, sur le même ordinateur, connecter différentes instances ensemble.

3.6.3 Les zombies en multijoueur

Après que Falvien ai fini l'implémentation des zombies, Corentin a dû revoir certaine partie du code pour qu'ils fonctionnent en multijoueur. En effet, pour une bonne expérience de jeu à plusieurs, il faut que le positon ainsi que la vie des zombies soit la même sur toutes les instances. Le plus logique pour obtenir une parfaite synchronisation est que ce soit le serveur qui s'occupe des zombies. Ainsi le serveur fait apparaitre les zombies, partage leurs positons à tous les clients et s'occupe des ragdoll et de l'argent qui tombe à la mort d'un zombie.

Le premier problème auquel Corentin a été confronté est la synchronisation des actions entre tous les joueurs et le serveur. En effet, lorsqu'un joueur modifiait l'état du jeu comme en tirant sur un zombie ce dernier était déconnecté. Pour régler ce problème, Corentin a mis en place des méthodes qui communique avec le serveur et les autres instances connectées au serveur pour qu'elles reçoivent l'information lors d'une modification.

3.6.4 Les problèmes rencontrés

Un problème, malheureusement, n'arrive jamais seul. Cette fois-ci, il venait de la synchronisation entre les joueurs. Lorsqu'un joueur se déplaçait, les autres joueurs ne voyaient aucun changement. C'est grâce au composant « Network Transform » appliqué sur le joueur que la synchronisation est possible. Malheureusement, comme un problème n'arrive jamais seul, un autre est apparu. La synchronisation entre les différents joueurs était très lente. En effet, lorsqu'un joueur se déplaçait l'autre joueurs le voyait se déplacer trop de temps après. Pour régler cela Corentin a optimisé les fonctions qui communiquent entre le serveur et les clients. Ainsi, comme le travail et moins centré sur le serveur, les déplacements deviennent un peu plus fluides.

3.6.5 Serveur

Jusqu'à la deuxième soutenance, le multijoueur fonctionnait en local, c'est-à-dire qu'on pouvait se connecter sur le même réseau pour jouer à plusieurs.

Le but souhaité était d'arriver à un système de multijoueur 100% en ligne. C'est à dire pas besoin d'être sur la même connexion pour se rejoindre. Cependant ceci nécessite un serveur pour accueillir les joueurs. Corentin a donc cherché un hébergeur, de préférence gratuit, pour héberger le temps des tests un serveur. Après

recherche, le choix du service AWS d'Amazon a été fait. C'est un service d'hébergement fiable avec 750 heure gratuite par mois pendant 1 an. Ce service permet de se connecter à un serveur dédié pour notre jeu.

Après avoir trouvé le serveur il fallait le configurer. Corentin à dû regarder de nombreux tutoriels car il n'avait aucune connaissance en réseau. Cependant, après avoir réussi à faire fonctionner le serveur, Corentin s'est rendu compte que le jeu se lançait avant même qu'un joueur se connecte. C'est pour cela, qu'il à dû revoir le script qui coordonne la partie pour faire en sorte que le jeu se lance uniquement lorsque tout les joueurs ont rejoint le serveur et sont prêt.

3.6.6 Lobby

Comme cité précédemment, Mirror possède nativement un HUD qui permet à un client d'héberger un serveur et jouer en même temps ou simplement de rejoindre une partie en ligne. Cependant, celui n'était pas très agréable à utiliser. C'est pourquoi, Corentin a ajouté une partie au menu qui reproduit le fonctionnement du HUD de Mirror. Le joueur a donc le choix entre jouer tout seul ou à plusieurs en entrant une adresse IP.

On avait l'intention de créer un système de « Room » qui permettait au joueur de se rejoindre avant d'arriver dans la prison. Cependant, du a un manque de temps et de la complexité à implémenter avec Mirror, l'idée a était abandonnée. Il sera peut-être implémenté plus tard après la fin du suivi du projet.

3.7 Site web

Le site internet de notre projet est un site de présentation du jeu, comme on peut en voir pour des jeux comme Valorant ou CoD Warzone. Il introduit les membres du studio, l'univers du jeu en présentant les ennemis, les armes, l'environnement dans lequel les joueurs devront se battre. Il permet de télécharger le jeu.

Flavien et Corentin ont travaillé sur le site web.

Le site est composé d'une page d'accueil qui permet de télécharger le jeu et d'accéder aux autres pages, parmi lesquelles on trouve une page présentant le jeu en générale, une présentant l'équipe, une qui présente le développement du jeu, ainsi qu'une page de contact.

Le site web est plutôt sobre et reprend l'univers du jeu.

Le développement du site a été un partie plus compliqué que ce que l'on imaginait. En effet, très peu d'entre nous avais des bases d'HTML et de CSS.

3.8 Marketing

3.8.1 Pochette du jeu

Une chose qui est aussi importante pour un jeu, c'est la pochette. Tom ayant déjà des connaissances dans le domaine du graphisme, c'est porté volontaire pour faire le design de la pochette de Zakharov Project. Tom a effectué le travail sur Photoshop et c'est basé sur la thématique du zombie comme le veut notre jeu.

FIGURE 8 – Aperçu de la pochette

3.8.2 Trailer

Une des raisons qui nous font aimer un jeu et le télécharger, c'est son trailer. Pour créer une ambiance. Entre musique, voix-off, visuels, le jeu vend du rêve et fait la promotion de son thème et atmosphère. Grâce au trailer, on a l'impression d'être plongé dans un film, une aventure, ce qui donne alors véritablement envie de se sentir transporté. Un bon trailer est donc essentiel pour l'aspect commerciale du jeu. Tom faisant régulièrement du montage vidéo, cette tache lui a été léguée.

Pour le trailer, Tom a utilisé trois logiciels : Unity, Sony Vegas Pro et Afetr Effect. Il voulait créer une ambiance atypique en plongeant les observateurs du trailer dans la vision directe du jouer. Pour se faire, Tom s'est servit du plugin d'animation de caméra pour faire plusieurs scènes dans la carte.

En parlant de la carte, Tom a rajouté tout une carte autour de celle de jeu afin de créer une réelle ambiance. Il a rajouté des montagnes, foret et lacs qui ont été fait grâce à l'outils Terrain disponible dans unity.

Pour finir, Tom a monté toutes les séquences sur Sony Vegas Pro et a rejouté quelques effet sur After Effect.

4 Bilan personnels

4.1 Corentin Belloni

La réalisation de ce projet a été pour moi très enrichissant d'un point de vu personnel, en développant mes compétences dans de nombreux domaines, mais aussi d'un point de vu social au sain d'un groupe. Je suis particulièrement fière du résultat obtenu même si certaine fonctionnalités imaginé n'a pas été implémenter.

La plus grande leçon que je retient de ce projet est l'autonomie. En effet, on nous demande de créer un jeux vidéo de A à Z sans aucune connaissance du processus de développement d'un jeu vidéo. Il a fallut que j'effectue mes recherches, que je me documente sur énormément de domaine auquel je n'avais jamais prêter attention. La gestion du multijoueur est la partie qui ma le plus appris même si elle a été très casse tête. En effet, je n'avais aucune connaissance du réseaux et des notions de serveur/client avant ce projet. De plus, mes connaissances en Unity et C# on nettement était améliorés grâce à ce projet.

Concernant le travail en groupe, je suis contant de notre coopération. Tout au long du projet la bonne ambiance était présente et cela motive d'autant plus pour avancé ensemble dans un projet de cet envergure. Chaque personne apportait des idées différentes et une façon de travailler inédite. Le travail de groupe est pour moi la meilleur des solutions car il permet de regrouper des personnes différentes sur un objectif précis avec chacune des compétences différentes. Aucun problème de groupe majeur durant ce projet, l'organisation était efficace.

Au final, cette expérience a été très enrichissantes pour ce qu'elle m'a apporté.

4.2 Tom Blancheton

Ayant fait la spécialisation Informatique et Science de l'ingénieur au lycée, j'avais déjà participé à un projet informatique. Ce projet était lui aussi à faire en groupe, avait lui aussi des deadlines, des rapports et beaucoup d'autres ressemblances avec notre projet de S2. Je n'ai donc pas découvert la notion de projet, de groupe et de gestion du temps de travail cette année.

Cependant, le projet réalisé cette année, Zakharov Project, était d'une tout autre nature et envergure que celui que j'avais réalisé en Terminale. Zakharov Project fut un projet beaucoup plus gros et qui me demanda beaucoup plus d'investissement. J'ai aussi pu apprendre beaucoup de choses grâce à celui-ci. La première chose étant bien sûr l'utilisation du logiciel Unity 3D. Bien que je pratiquais déjà la modélisation 3D avec des logiciels comme Cinéma 4D et Blender, j'ai beaucoup appris grâce à Unity. Cependant la chose qui m'aura le plus plu et le plus marqué durant ce projet fut d'apprendre à utiliser et à manipuler tout ce qui tourne autour des menus et des boutons.

J'ai aussi particulièrement aimé la phase de réflexion pour trouver une solution au problème de compatibilité entre mon programme principal et Unity 3D. Pour finir j'ai adoré la réalisation du trailer et de la pochette du jeu car les montages vidéos et/ou photo son une de mes passions. Apprendre la gestions des caméras et leurs animation étaient très enrichissant pour moi !

Grace au projet, surtout au début du projet, j'ai appris à travailler avec des personnes n'ayant pas forcément les mêmes envies et visions à propos du projet (contrairement au projet réalisé en Terminale). Je ne regrette absolument pas d'avoir était membre du groupe de Zakharov Project.

4.3 Lucas Cizeron

Ce projet a été très instructif en de nombreux points, depuis toujours intéressé par la production de jeux vidéo, il était l'occasion pour moi de pratiquer ce hobby auquel je n'avais plus de temps à accorder.

Le projet m'a appris de nombreuses compétences humaines et de communication dont certaines sont essentielles comme le travail de groupe ou l'organisation en équipe, des aspects qui me paraissent très importants pour une carrière d'ingénieur. Cela permet de se rendre compte de ce que le travail de groupe peut nous apporter. En effet on a certaines difficultés rajoutées lors d'un travail à plusieurs comme par exemple le partage de fichiers et de code via git qui peut être assez compliqué à gérer.

En revanche, avoir des coéquipiers peut aussi être un gain de temps, l'entraide et la mise en commun des connaissances de chacun est un véritable atout et permet de se débloquer de certaines situations handicapantes.

Le projet nous a également permis d'acquérir des compétences techniques, la création de jeux évidemment est un domaine que l'on a beaucoup travaillé avec ce projet, mais le plus important est finalement tout ce qui vient à coté de cette création. On parle ici de compétences en C mais aussi de l'utilisation de logiciels tiers en tout genre et dont la maîtrise peut servir à tout ingénieur, peu importe son domaine.

C'est finalement toutes ces capacités secondaires engendrés par le projet qui sont réellement intéressantes à acquérir.

4.4 Flavien Geoffray

Le Projet Zakharov à été pour moi très enrichissant. Il m'a permis d'acquérir de nombreuses compétences techniques mais aussi de découvrir la réalisation complète d'un jeu vidéo. Souhaitant travailler plus tard dans le monde du jeu vidéo, j'ai beaucoup aimé ce projet, que l'on a pris très au sérieux. D'un point de vue global, je suis très satisfait du travail que nous avons accomplis. La cohésion et l'organisation au sein de notre groupe à aussi rendu le projet plus enrichissant.

Pour la première fois, je travaillais sur un projet qui ne se limitait pas à la programmation. J'ai beaucoup aimé suivre un cahier des charges, travailler en équipe, écrire des rapports, trouver des solutions, perfectionner le jeu, des choses qui ont permis de rendre le projet sérieux et concret à mes yeux.

Le fait de n'avoir aucune connaissance et compétences en développement de jeu ne ma pas forcément dérangé et à même rendu les choses intéressantes puisque qu'il a impliqué un véritable investissement. J'ai du me documenter, regarder des vidéos, et beaucoup apprendre par moi même. Cela m'a permis d'acquérir beaucoup de connaissance techniques dans l'utilisation de logiciel comme Unity ou Blender pour la modélisation et quelques animations notamment.

Ce projet m'a également permis de perfectionner ma manière de coder en C et d'acquérir de l'expérience dans le travail d'équipe. J'ai aussi pu faire un premier pas dans l'intelligence artificielle au sein des jeux vidéos, puisque mon rôle dans ce projet à principalement consisté à mettre en place le système d'ennemi.

Dans l'ensemble ce projet à été pour moi une expérience très enrichissante. Participer ou créer un jeu vidéo de A à Z était quelques choses que je souhaitais faire depuis longtemps. C'est désormais fait. Je suis très satisfait de notre jeu et du travail que nous avons fait. En revanche, j'aurais aimé avoir plus de temps. En effet, comme dans tout projet nous avons rencontré de nombreux problèmes, ce qui nous à empêcher de mettre en place quelques points du cahier des charges. Néanmoins, il est fort probable que je travaille a nouveau sur le jeu à titre personnel, pour améliorer et/ou compléter le Projet Zakharov.

5 Conclusion

La réalisation de Zakharov : Undead Prison est une expérience enrichissante dans tous les domaines pour chacun d'entre nous. Durant le développement, nous avons dû faire face à de nombreux problèmes de réalisation, de bugs ou même de désaccord. Malgré ça, nous avons réussi à avancer dans le développement de ce projet qui nous tenait tous les quatre à coeur.

Lors du développement, le plus grand défi était de regrouper toutes nos versions ensemble. En effet, lors de la fusion de nombreux conflits apparaissaient et était très long à réparer. Cependant, avec le temps et l'expérience acquise, ce type d'erreur était de moins en moins présent.

Pour l'organisation, elle sait faite naturellement grâce aux différentes compétences de chacune. Avant le début du développement on sait tous mis d'accord pour chacun travaillent sur une partie qui nous plaisait. Ainsi, le développement fût rapide et amusant pour la plupart du temps.

Finalement, nous sommes arrivé au bout de ces six mois de développement, chacun en ressort grandit avec de toutes nouvelles connaissances et une énorme satisfaction envers nos accomplissements. Nous sommes fières de vous présenter notre jeu Zakharov : Undead Prison.

6 Annexes

6.1 Webographie

- **Mirror :** https ://assetstore.unity.com/packages/tools/network/mirror-129321
- **Mixamo :** https ://www.mixamo.com/
- **Quixel :** https ://quixel.com/
- **Unreal Engine :** https ://www.unrealengine.com/en-US
- **Photoshop :** https ://www.adobe.com/fr/products/photoshop.html

- **Freesound :** https ://freesound.org/
- **Blender :** https ://www.blender.org/
- **Unity :** https ://unity.com/fr
- **Illustrator :** https ://www.adobe.com/fr/products/illustrator.html

- **GitHub :** https ://github.com/
- **Overleaf :** https ://fr.overleaf.com/
- **Sketchfab :** https ://sketchfab.com/

6.2 Table des illustrations

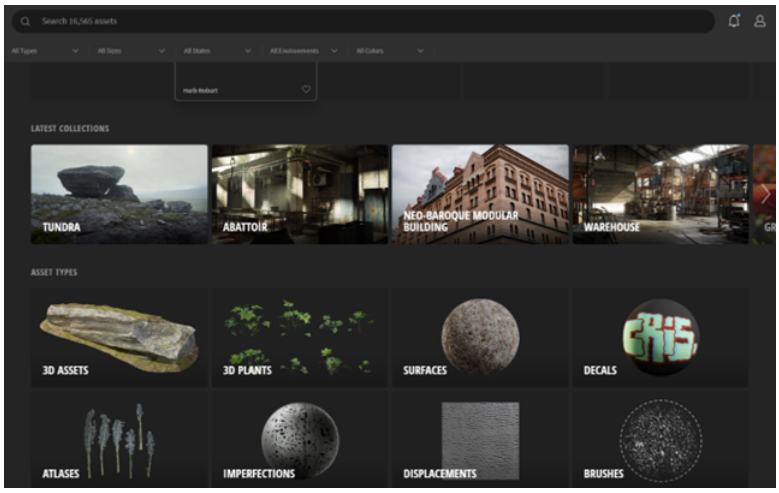

FIGURE 9 – Aperçu des différents types de ressources que propose Quixel

FIGURE 10 – Schéma de la carte

FIGURE 11 – Simulation physique d'un grand nombre de zombies

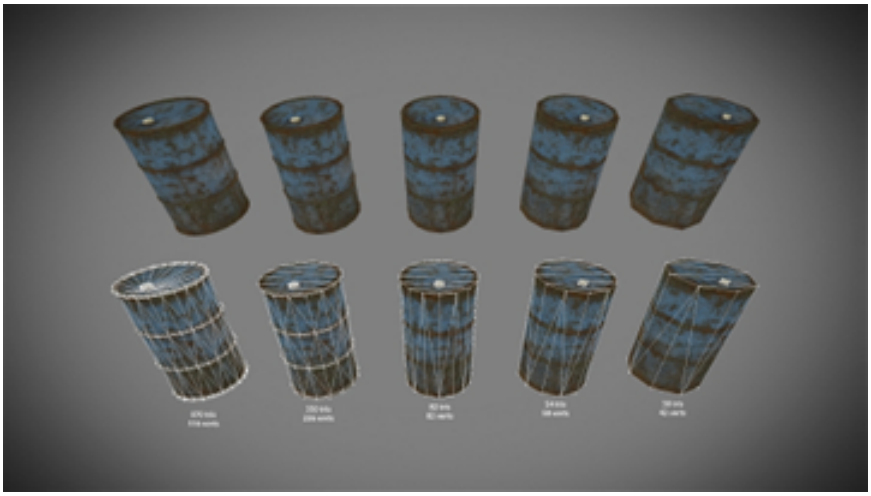

FIGURE 12 – Différents LOD d'un même modèle 3D avec leur nombre de triangles

FIGURE 13 – Pistolet Makarov

FIGURE 14 – Pistolet-mitrailleur : le PPSh-41

FIGURE 15 – Carabine semi-automatique : le Sks