

Rapport de soutenance

Tales of Zyfron

EPITA 2028: Projet S2

Groupe CAMAL INC



RAMSTEIN Antoine

MULLER Célian

HOLLAND Miles

MEYER Aurélien

ZHOU Laurent

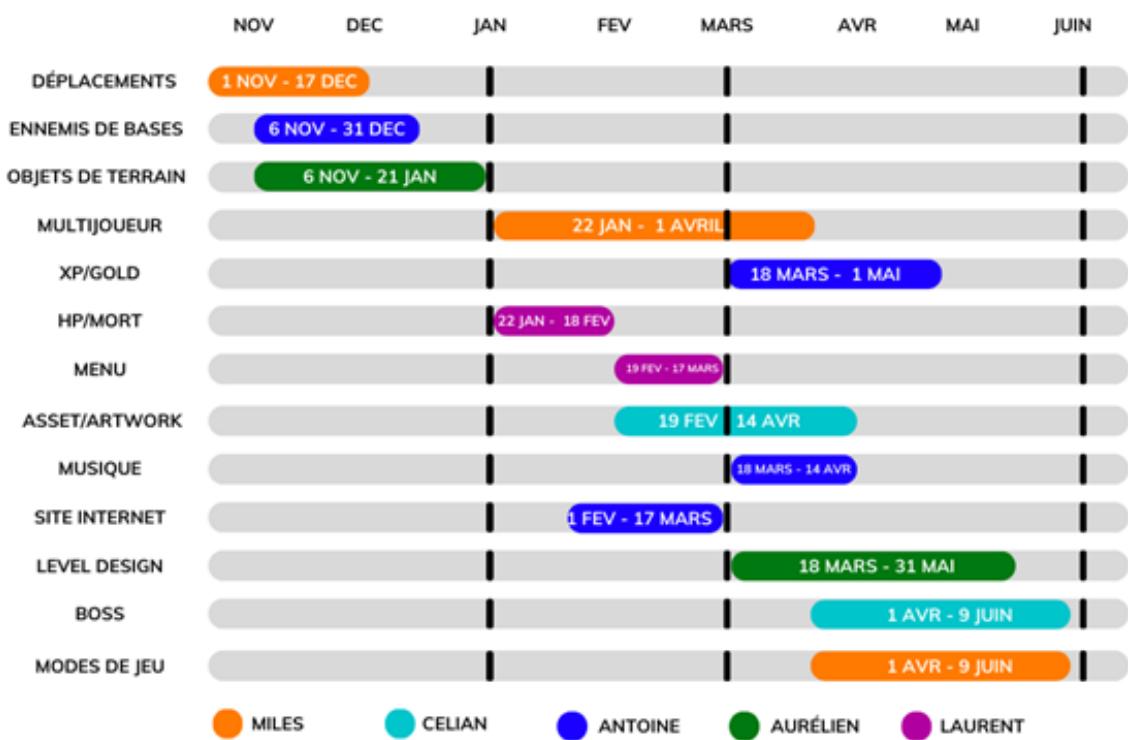
TABLE DES MATIÈRES

1	Introduction	2
1.1	Planning - Diagramme de Gantt	2
1.2	Présentation des Tâches	3
2	Descriptption des tâches réalisées	4
2.1	Déplacement du Player Controller	4
2.1.1	Saut	5
2.1.2	Sprint	5
2.1.3	S'accroupir	5
2.1.4	Glissade	5
2.1.5	Wall run	6
2.1.6	Du déplacement à l'animation	6
2.2	Ennemis	8
2.3	Multijoueur	13
2.4	Objets de terrain	14
2.4.1	Plateformes	14
2.4.2	Portes	15
2.4.3	Objets divers	16
2.4.4	Problèmes rencontrés	17
2.4.5	Monde SandBox	18
2.5	Asset / Artwork	20
2.5.1	Utilisation du site internet Mixamo	20
2.5.2	Problème de texture sur le modèle 3D d'un ennemi	20
2.6	Système d'HP / mort et de combat sur des Game Object	22
2.7	Menu	24
2.8	Problèmes / Solutions à envisager	24
3	Tâche à réaliser lors de la soutenance finale	25
3.1	Créer un système d'XP/Gold	25
3.2	Amélioration des Assets	25
3.3	Musique	25
3.4	Level design	26
4	Conclusion	27

1 INTRODUCTION

1.1 Planning - Diagramme de Gantt

TALES OF ZIFRONT



1.2 Présentation des Tâches

Le rapport de stage présente sur une vingtaine de pages ce qui a été réalisé depuis la validation du cahier des charges. Il doit faire le bilan de ce qui a été fait (avances et/ou retards sur planning), par qui et présenter ce qui doit être fait pour la prochaine soutenance, à savoir la soutenance finale. L'objectif de cette première soutenance est de présenter les différentes fonctionnalités que l'équipe CamalInc à réaliser après la validation du cahier des charges, à savoir des déplacements, des ennemis, des objets de terrain, un multijoueur fonctionnel, un site internet, un menu ainsi qu'un système d'HP et de mort. La réalisation de ces tâches nous permettra d'avancer sereinement et dans de bonnes conditions sur le projet. De plus, il a pour but d'offrir une expérience unique, permettant aux joueurs débutants comme avancés de s'épanouir dans un nouvel univers. Nous voulons offrir à chaque joueur la sensation de vivre sa propre aventure et d'y faire ses propres choix.

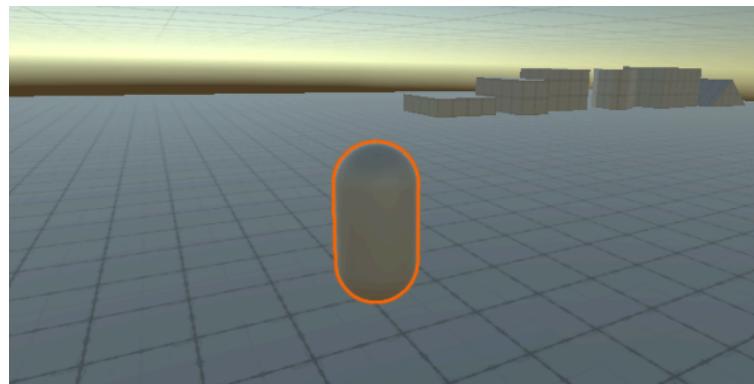
2 DESCRIPTPION DES TÂCHES RÉALISÉES

2.1 Déplacement du Player Controller

Tâche réalisée par Miles HOLLAND et Laurent ZHOU

Dans le cadre de ce projet axé sur le platforming, nous accordons une priorité absolue à la conception des déplacements du personnage. Notre objectif principal est de garantir une expérience de jeu fluide, où le joueur se sent à l'aise et en contrôle total de ses mouvements. En effet, des déplacements fluides sont essentiels pour offrir une jouabilité immersive et captivante.

La création du personnage est une étape fondamentale avant d'implémenter les fonctionnalités de mouvement. Cela implique la mise en place d'un objet vide (empty game object) auquel une capsule est attachée pour représenter le personnage. De plus, une caméra est ajoutée pour permettre une vue à la première personne. Cette caméra est équipée d'un script pour gérer le mouvement de la vue.



Les déplacements de base sont gérés en utilisant la fonction Input.GetAxisRaw, qui permet de récupérer les entrées de déplacement du joueur. En utilisant ces entrées, le personnage est déplacé en ajustant les vecteurs en X et en Z, ce qui permet de contrôler sa position sur le plan horizontal. Grâce à cette implémentation, le joueur peut contrôler le mouvement du personnage de manière intuitive, en utilisant les touches directionnelles ou les touches W, A, S, D pour se déplacer dans différentes directions

2.1.1 Saut

Les sauts jouent un rôle essentiel dans notre jeu de platforming en permettant au joueur de franchir des obstacles et d'explorer l'environnement en trois dimensions. L'implémentation des sauts permet au joueur de contrôler le moment et la hauteur de chaque saut, offrant ainsi une expérience de jeu dynamique et stimulante.

2.1.2 Sprint

La fluidité est une priorité, c'est pourquoi nous avons inclus la fonctionnalité de sprint. En appuyant sur une touche spécifique, le joueur peut choisir de marcher ou de sprinter, ce qui ajuste la vitesse de déplacement du personnage en conséquence. Cette fonctionnalité permet au joueur de s'adapter rapidement aux différents défis du jeu.

2.1.3 S'accroupir

L'action de s'accroupir est devenue une fonctionnalité importante dans de nombreux jeux. Bien qu'elle puisse être utilisée pour des raisons pratiques, comme se cacher derrière des obstacles, elle est souvent utilisée de manière ludique pour taquiner les autres joueurs ou pour ajouter une dimension stratégique au gameplay. Dans notre jeu, les joueurs peuvent s'accroupir pour éviter les obstacles bas, se cacher des ennemis ou simplement pour ajouter une touche de réalisme à leur expérience de jeu.

2.1.4 Glissade

La glissade constitue une fonctionnalité essentielle dans notre jeu, offrant au joueur la possibilité de se déplacer rapidement le long de rampes ou d'esquiver des projectiles. Cette fonctionnalité contribue grandement à la sensation de fluidité et de dynamisme de notre jeu, en permettant aux joueurs de naviguer de manière agile et réactive à travers l'environnement. En offrant cette capacité de glissade, nous améliorons l'expérience de jeu globale en donnant aux joueurs des moyens supplémentaires pour interagir avec le monde du jeu et pour surmonter les défis de manière créative et engageante.

2.1.5 Wall run

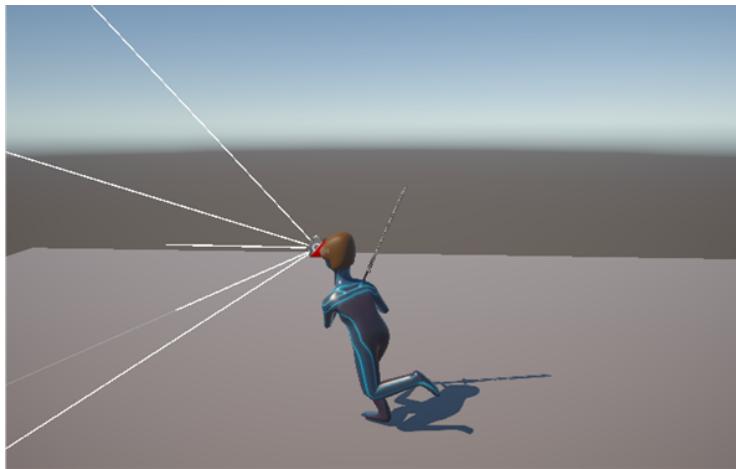
Cette fonctionnalité joue un rôle crucial dans l'enrichissement de l'expérience de jeu. En permettant aux joueurs de courir le long des murs, nous leur offrons une nouvelle dimension de mouvement et d'interaction avec l'environnement. Cela ouvre la porte à une variété de possibilités de gameplay, notamment des séquences de plateforme complexes, des itinéraires alternatifs dans les niveaux et des stratégies de combat innovantes.

De la marche de base au sprint, en passant par les sauts, la glissade et les courses sur les murs, chaque fonctionnalité contribue à enrichir le gameplay et à offrir aux joueurs une variété d'options de mouvements. Ces fonctionnalités ne servent pas seulement à franchir des obstacles, mais elles ouvrent également de nouvelles perspectives en termes de conception de niveaux, de stratégies de jeu et d'interactions avec l'environnement. En fin de compte, des déplacements fluides et agréables sont essentiels pour créer une expérience de jeu mémorable et satisfaisante.

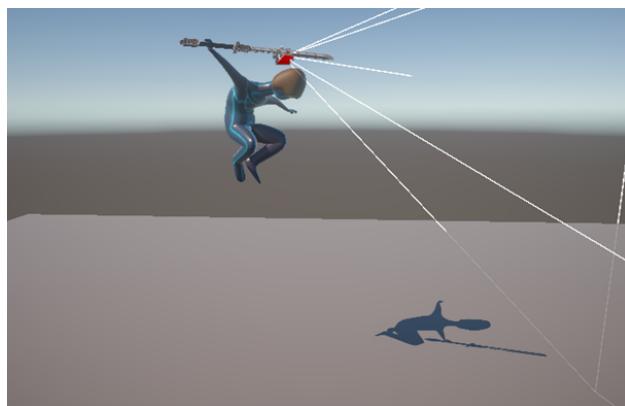
2.1.6 Du déplacement à l'animation

L'objet Player Controller (PC) est composé de plusieurs sous-objets, principalement l'objet physique et l'objet visuel. Il est important que l'animation soit incrémentée en tant que classe à part, attachée au sous-objet visuel, mais qui communique avec la classe qui gère la partie logique et physique du PC. Organiser le PC comme cela permet de faciliter la synchronisation lors du passage en multijoueur, et en cas de besoin, de modifier le script logique sans devoir tout réimplémenter.

Lorsque la touche d'action "Avancez" est sélectionnée, la classe logique attachée au PC déplace le corps physique vers là où le point de la caméra se situe . Par ailleurs, Une méthode publique vérifie simplement si le corps est aux mêmes coordonnées qu'aux frames précédentes, et renvoie "true" si c'est le cas et "false" dans le cas contraire. Cette méthode est appelée dans l'objet d'animation et active l'animation pour avancer quand elle reçoit "true".



Pour sauter, on implémente une méthode qui vérifie si l'objet physique touche le sol ou non. Enfin, pour attaquer, la méthode ne renvoie pas true ou false, mais active simplement l'animation de l'attaque.

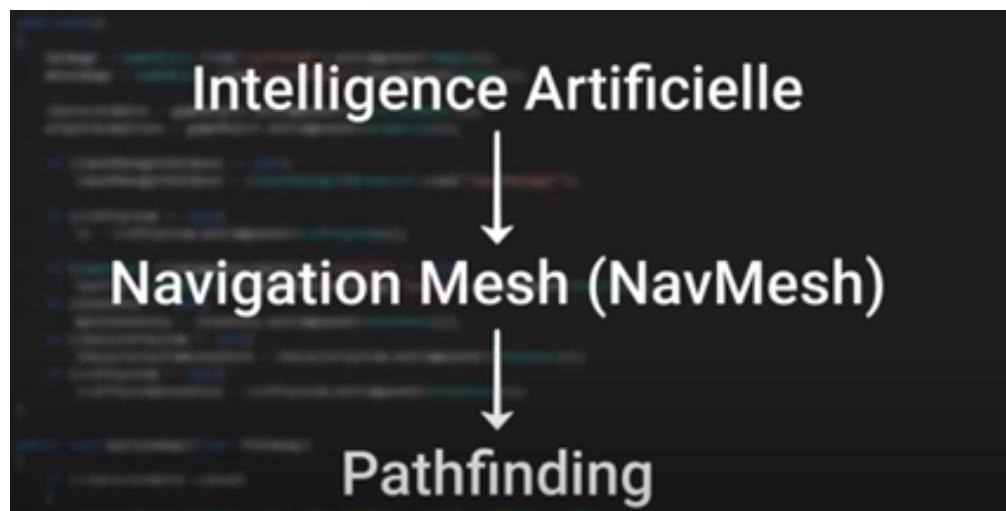


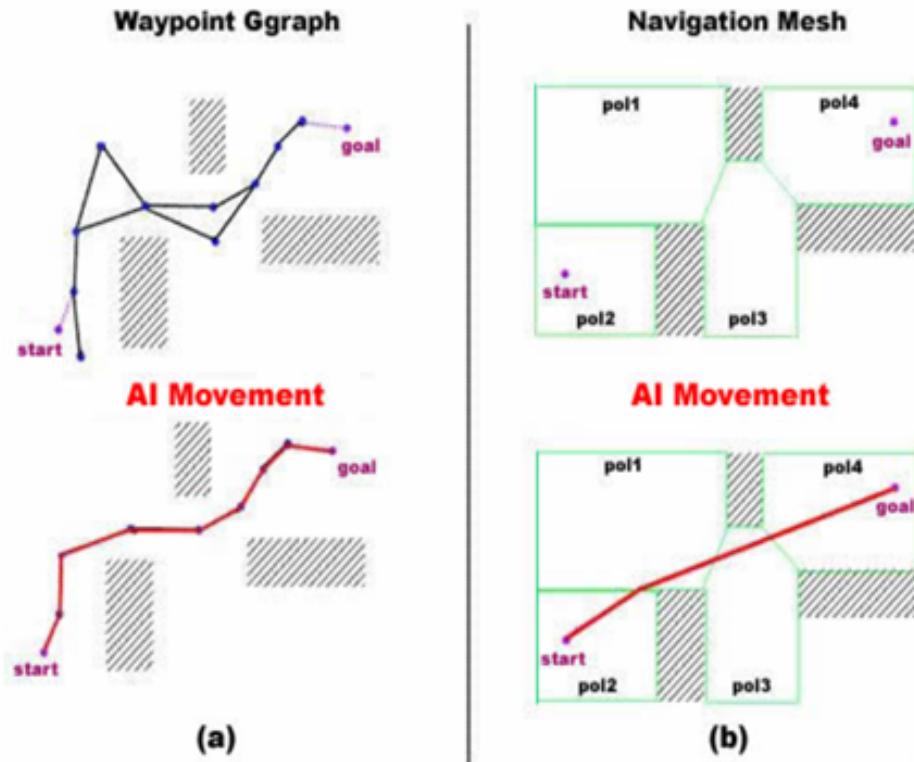
La classe d'animation ne peut pas animer la partie visuelle du PC en lui-même. Unity a un animateur qui est un outil très puissant, permettant de gérer les animations de manière intuitive sous forme de schéma. L'animateur est également attaché à l'objet visuel, ce qui permet au script de communiquer avec lui à l'aide de tags. Un tag prend comme valeur un string, il faut donc faire attention à appeler la bonne fonction avec la bonne orthographe, par exemple si on écrit "Isgrounded" à la place de "IsGrounded" l'animator ne comprendra pas et ne fera tout simplement pas l'action. Ce problème a été récurrent, lorsque l'on a implémenté les animations des ennemis mais aussi du Player Controller (joueur).

2.2 Ennemis

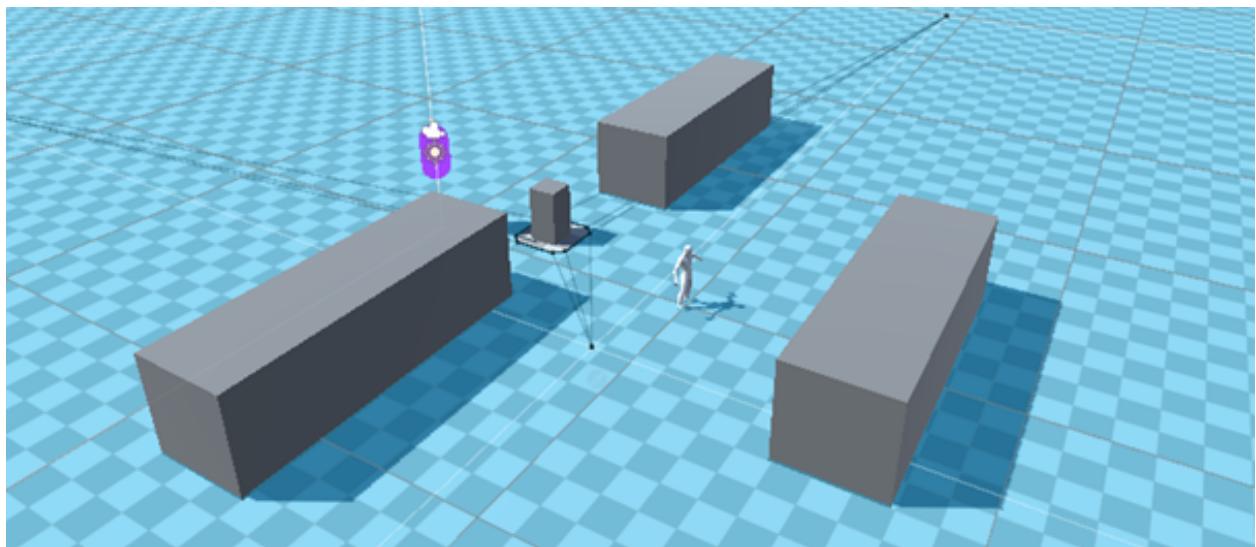
Tâche réalisée par Célian MULLER et Antoine RAMSTEIN

Pour reproduire le patronne d'un ennemi, nous avons utilisé le Package IA Navigation, notamment NavMesh. NavMesh (Navigation Mesh), correspond à une structure de données utilisé en intelligence artificielle permettant de représenter les zones d'un environnement 3D. Elle permet donc à une intelligence artificielle de se déplacer dans un environnements malgré des contraintes physiques (mur, obstacle, puit, ...). Sa principale différence est qu'elle est spatiale et préparée pour des espaces spatiaux continus, tandis que la recherche de chemin est discrète et présupposée fonctionner avec des coordonnées discrètes fixes.

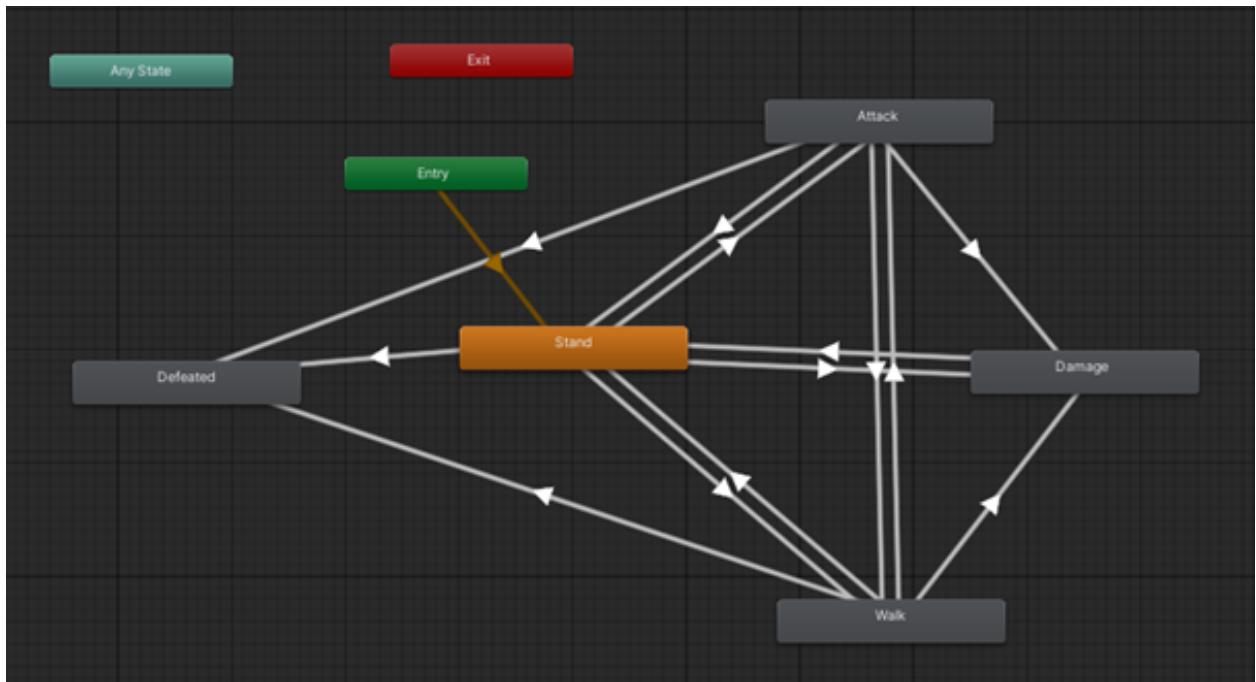




Elle permet d'optimiser la recherche de chemin. En précalculant et en stockant des informations sur les zones praticables, les jeux peuvent réduire considérablement la charge de calcul associée à la recherche de chemin en temps réel, améliorant ainsi les performances globales. Ils peuvent prendre en compte des environnements de forme irrégulière, des pentes et des terrains variables, permettant ainsi des mouvements de personnages plus réalistes et plus fluides. De plus, son utilisation permet d'éviter les collisions. Les Navmeshes contribuent à des stratégies efficaces d'évitement des collisions. Les personnages peuvent contourner intelligemment les obstacles, ajuster leur trajectoire en temps réel pour éviter les collisions et créer une expérience de jeu plus immersive. Grâce à son utilisation, il est donc plus facile d'intégrer une intelligence artificielle pour définir le comportement d'un personnage non jouable (PNJ).



Le prototype contient les actions de bases à savoir attaquer (Attack), recevoir des dégâts de la part du Player (Damage), marcher (Walk) et mourir (Defeated).

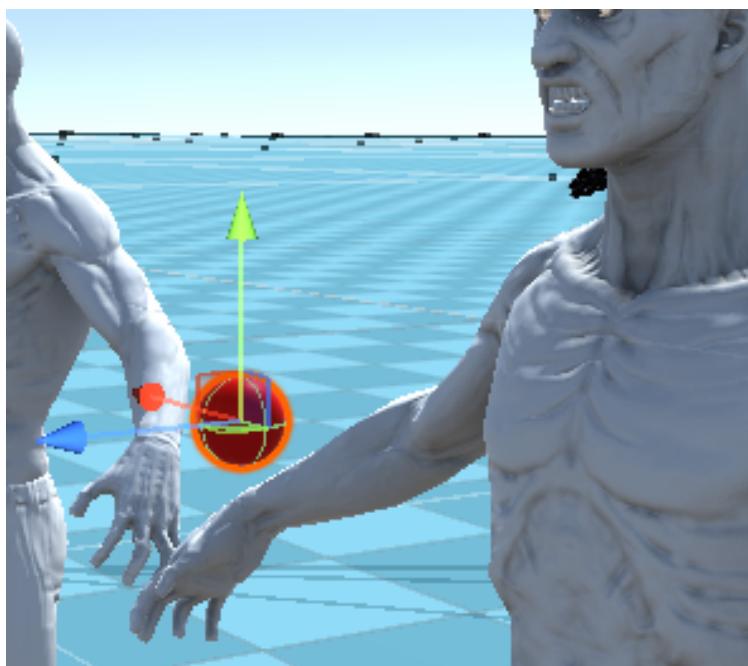


Lorsque le Player rentre dans le champ de vision de l'ennemi, le prototype se met à l'attaquer, et lorsqu'il s'éloigne assez de son périmètre, il se stop (animation Stand), attendant que le

Player se trouve dans champ d'actions pour réaliser son script.

Nous avons mis en place un système d'attaque corps-à-corps pour l'ennemi. En effet, il fallait assigner un cube 3D invisible dans la main de l'ennemi qui par l'intermédiaire d'un script, nous permet d'infliger des dégâts au Player. Lorsque l'ennemi est assez proche du Player (GameObject), les points de vie de celui-ci diminuent lorsque le cube 3D rentre en collision avec le GameObject.

Nous avons également implémenté un ennemi qui tire des projectiles à longue distance.

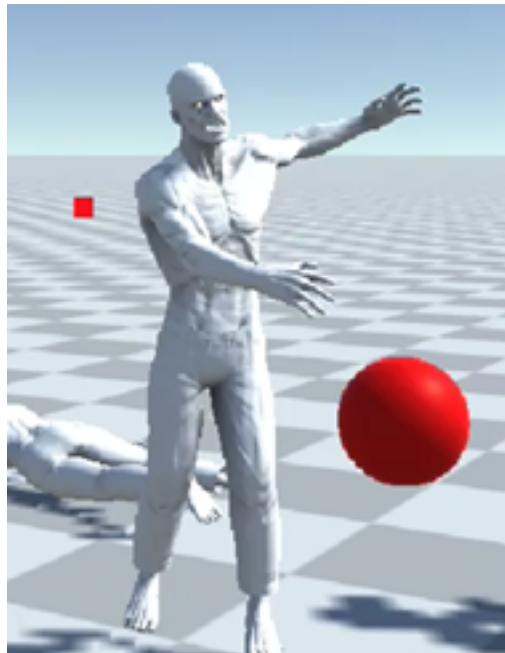


La balle rouge représente le projectile, un Game Object avec un material et un RigidBody. Il suffit de faire partir notre Game Object projectile d'un empty game object au modèle du zombie qu'on a appelé SpawnPoint, le point d'apparition du projectile.

Il faut déclarer différents attributs qui permettent de faire fonctionner notre projectile :

- Un NavMeshAgent qui représente l'IA ennemi
- Un Transform qui représente le joueur
- Un float bulletTime qui récupère le temps écoulé entre les tirs
- Un float timer, un compteur qui déterminera la cadence de tir
- un GameObject enemyBullet le préfabriqué de la balle rouge
- un float enemySpeed la vitesse de la balle rouge

L'objectif est donc de projeter une sphère rouge, représentant une balle, de l'ennemi à une certaine distance avec un cooldown entre chaque tir. On détruit le projectile après un délai de 1 seconde afin de ne pas surcharger le serveur avec trop d'entités sur la map. Cela pourrait engendrer une perte de FPS (Image par seconde) et rendre le jeu moins fluide pour les joueurs. Si le projectile touche le joueur (Player Controller), il perdra de la vie, de la même façon que l'ennemi corps-à-corps. Cet ennemi est immobile est ne s'active que lorsque le joueur rentre dans le champ de vision du prototype. De la même façon que l'ennemi corps-à-corps, il n'attaque plus lorsque le joueur est trop loin (un obstacle, comme un mur, empêche l'ennemi de voir le joueur et donc de tirer).



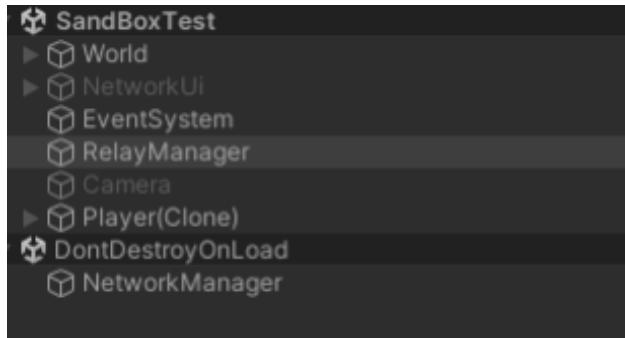
2.3 Multijoueur

Tâche réalisée par Miles HOLLAND et Laurent ZHOU

Sur Unity, un des moyens les plus simple pour implémenter le multijoueur est d'utiliser l'outil Netcode. Une fois installé et implémenté, il suffit de tout synchroniser sur le même serveur. Par ailleurs, nous avons tenter de créer un multijoueur en utilisant FishNet mais sans aboutir.

L'implémentation du multijoueur représente un défi technique important. Nous avons donc choisi de commencer par mettre en place le multijoueur en mode local. Avec Netcode for GameObjects, cela implique la création d'un Network Manager, ainsi que l'ajout de composants tels que Network Object, Network Transform et Network Rigidbody sur les objets du joueur. Cependant, nous avons rencontré un problème lors du déplacement des joueurs : les mouvements affectaient simultanément tous les joueurs présents. Après analyse, nous avons identifié que cela était dû à la gestion de l'attribut "owner" incorrecte. En ajustant notre code pour vérifier correctement le propriétaire d'un objet avant de le déplacer, nous avons pu résoudre ce problème.

Concernant le multijoueur en ligne, nous avons choisi d'utiliser RELAY, un outil développé par Unity pour faciliter la connexion entre les joueurs distants. Pour mettre en place cette fonctionnalité, nous avons ajouté un GameObject vide nommé RelayManager. Le script relié à RelayManager est responsable de la gestion de la création et de la connexion à des relais pour le multijoueur en ligne utilisant Unity Relay. Le script utilise Unity Services pour initialiser les services nécessaires au démarrage du jeu, notamment l'initialisation du service d'authentification. Une fois que l'authentification anonyme est réussie, le joueur est connecté et son ID est enregistré.



Le problème le plus récurrent est de synchroniser les joueurs entre eux sur le même serveur. Pour se faire, nous avons décidé de donner l'autorité aux clients, c'est-à-dire que chaque client exécute son script, ainsi le serveur ne prend pas de décisions. De plus, cette méthode ne prend pas en compte la sécurité du serveur, une personne peut très bien modifier à sa guise les valeurs du joueur.

Chaque entité hostile doit donc être synchronisée, ainsi que chaque autre objet qui peut interagir avec les joueurs (Player Controller).

En résumé, bien que l'implémentation du multijoueur présente des défis, nous avons progressé avec succès dans la mise en place du multijoueur en local et en ligne grâce à l'utilisation de Netcode for GameObjects et de RELAY.

2.4 Objets de terrain

Tâche réalisée par Aurélien MEYER

2.4.1 Plateformes

Tout d'abord différents types de plateformes ont été réalisés :

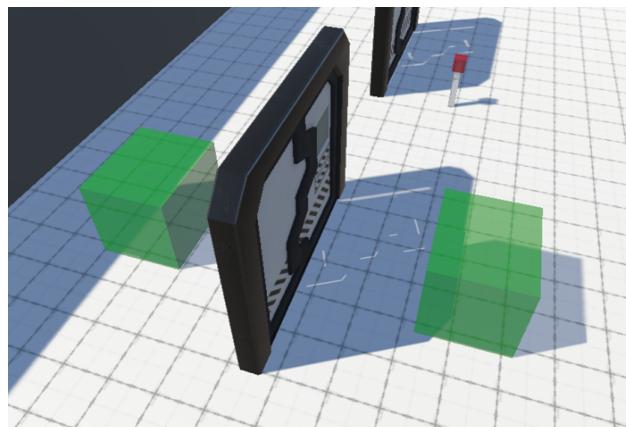
- Plateforme en mouvements basiques, c'est-à-dire qu'avec une animation, la plateforme peut faire un mouvement défini. Avec l'ajout d'un script qui détecte la présence d'un joueur, le soude à la plateforme pour que le joueur se déplace avec la plateforme, dans le cas échéant, la plateforme bougera sans le joueur.
- Plateformes en mouvement destructible, ces plateformes ont le même fonctionnement que les premières, mais à un certain moment ces plateformes se détruisent.
- Plateforme statique destructible, au contraire des précédentes ces objets ne bougent pas,

mais se détruisent après un certain temps une fois que le joueur est détecté sur la plateforme. Le but ici est d'enchaîner les sauts rapidement pour ajouter de la rapidité dans le gameplay. A ajouter que dans les cas des plateformes destructibles il y a un système de régénération de la plateforme pour qu'en cas d'échec le joueur puisse recommencer.

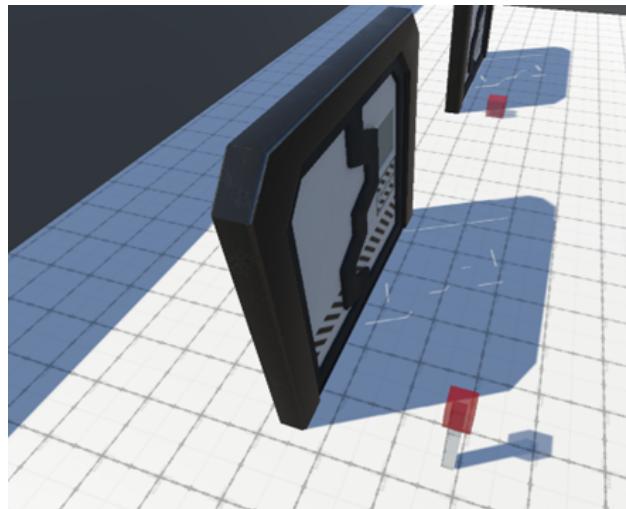
2.4.2 Portes

Trois portes coulissantes avec des états d'actions différents ont été créées :

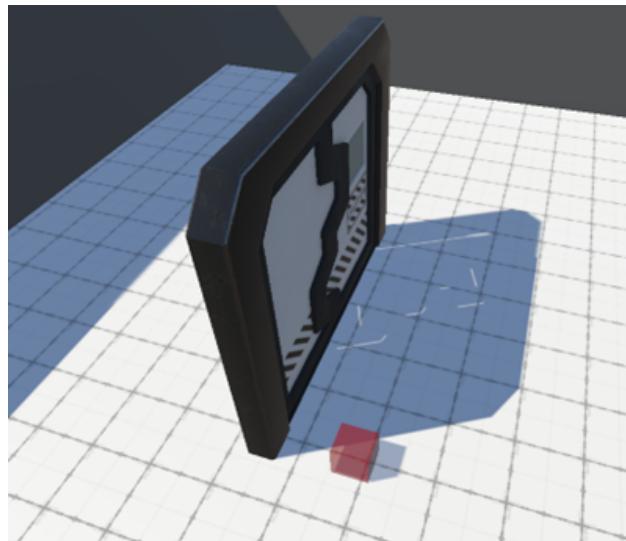
- Porte Trigger, ces portes s'ouvrent avec un trigger (blocs verts) qui détecte la présence d'un joueur et qui ouvre la porte si elle est fermée, et la ferme dans le cas inverse. Ces portes peuvent être utilisées pour, par exemple, accéder à un nouveau niveau.



- Porte Bouton, ces portes s'ouvrent avec un bouton (Game Object, Cube 3D rouge) qui détecte si un joueur a appuyé sur un bouton ou non. La détection du bouton se fait par un Raycast que le joueur envoie sur le bouton. Pour se faire, il y a deux conditions, le joueur doit être suffisamment proche, doit aussi regarder le bouton. Une fois appuyer, la porte s'ouvre ou se referme.



- Porte Clé, ces portes s'ouvrent avec une clé qu'il faut ramasser et faire interagir avec la porte. Elle ne s'ouvre que si le joueur à la clé en sa possession et qu'il interagit avec la porte à l'aide d'un Raycast.

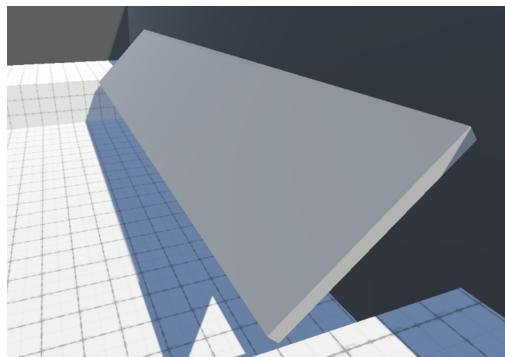


2.4.3 Objets divers

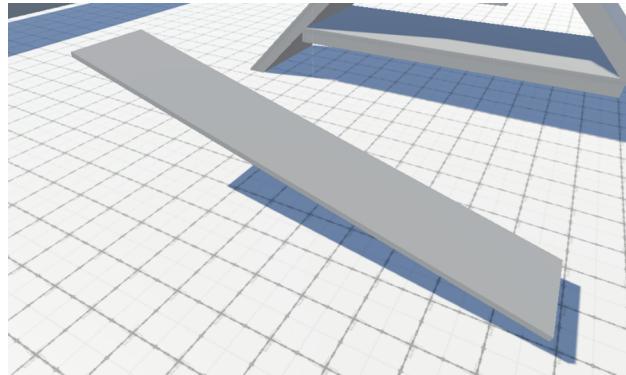
Ce sont des objets très simples, qui demandent peu de connaissances, mais sont vitales pour le gameplay.

- Le premier est une simple rampe, qui grâce à la mécanique de glissade, permettra au joueur

de se déplacer de plateforme en plateforme pour ajouter de la rapidité dans le gameplay. Il sera aussi possible de changer l'orientation de la plaque pour glisser sur les murs.



- Le second est une balançoire à bascule. Avec une charnière placée au milieu de celui-ci, le joueur devra utiliser l'équilibre et la rapidité pour passer d'un côté à l'autre.

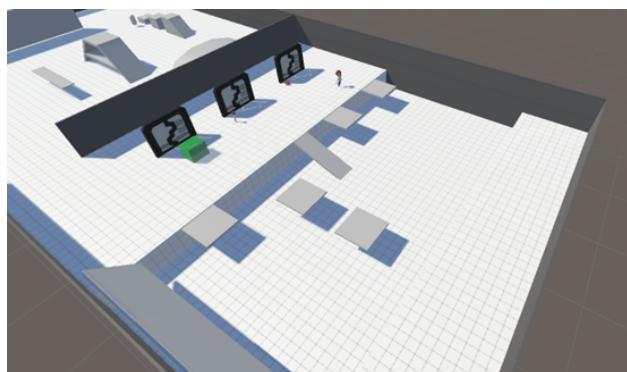
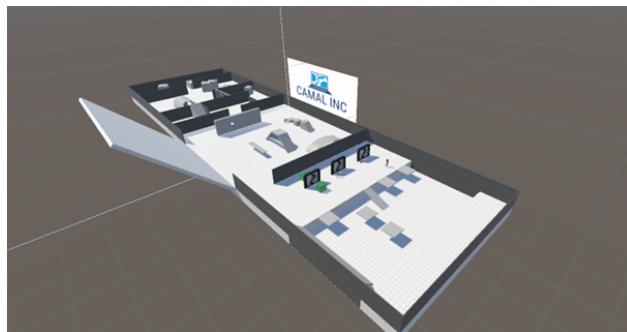


2.4.4 Problèmes rencontrés

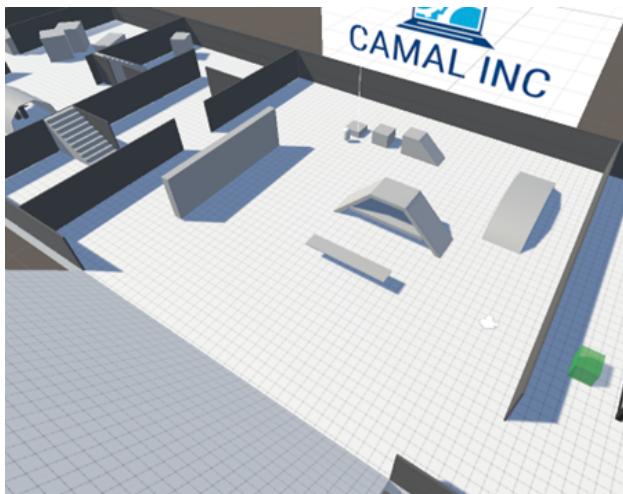
Le problème majeur rencontré lors de la création des objets de terrain sont les triggers pour le multijoueur. Pour un jeu solo, il suffit de mettre l'objet dans une variable du script du trigger pour qu'il le détecte. Or, dans le cas du multijoueur, il faut qu'il détecte 4 joueurs différents . Il fallait donc créer différents NetworkObject pour que les trigger ne détectent plus qu'un seul objet.

2.4.5 Monde SandBox

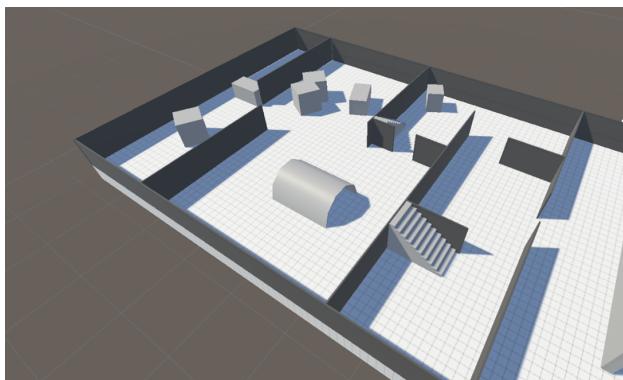
Dans le but de regrouper les travaux de chaque membre de l'équipe dans un seul et même endroit, nous avons réalisé un premier niveau "SandBox", c'est-à-dire bac à sable. Ce premier niveau nous aidera à implémenter et tester de nouveaux ajouts, c'est pour cela que le niveau est divisé en trois parties.



Création, Tests d'objets de terrain



Partie déplacement du joueur



Tests des ennemis et du Player Controller

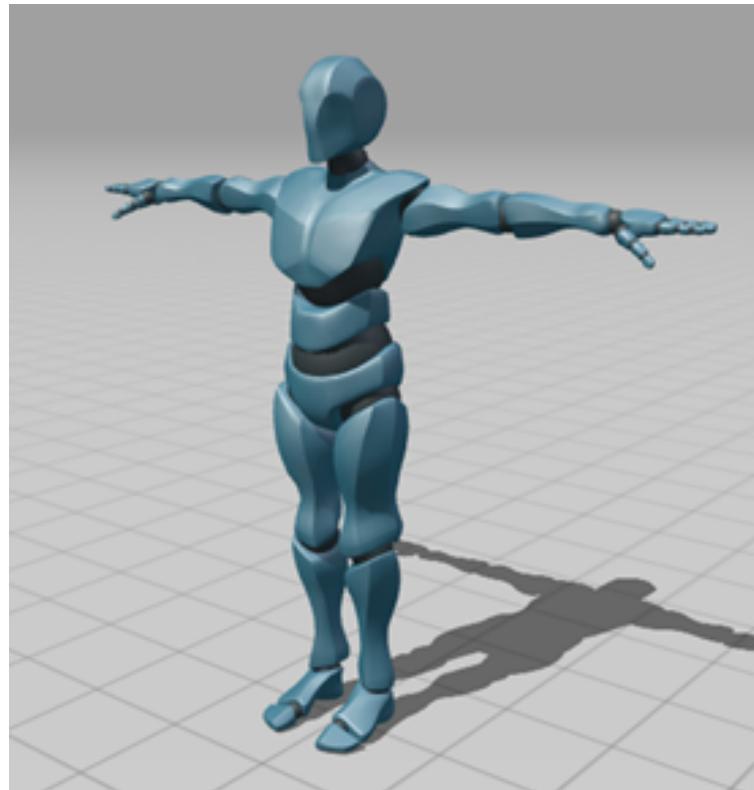
A ajouter que le niveau est agrandissable pour effectuer plus de tests.

2.5 Asset / Artwork

Tâche réalisée par Antoine Ramstein et Célian MULLER

2.5.1 Utilisation du site internet Mixamo

Mixamo propose en grand nombre d'assets que ce soient des personnages déjà Riggeés ou même des animations dynamiques. Dans un premier temps, le but est de démontrer que les éléments essentiels de notre jeu fonctionnent correctement. Ainsi les assets n'étant pas notre priorité, nous avons choisi de rester sur un design assez simple et facile à manipuler.



2.5.2 Problème de texture sur le modèle 3D d'un ennemi

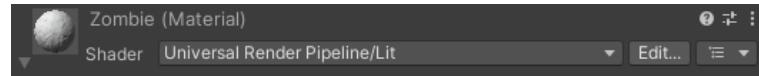
Lors de l'importation d'un modèle 3D pour un ennemi de référence, les textures de celui-ci apparaissaient roses en raison d'un problème de rendu.



En recherchant une solution à notre problème, nous avons remarqué que le souci provenait du shader (nuanceur). Celui-ci était réglé sur le paramètre standard et il se peut que celui-ci n'était pas compatible avec les textures et les matériaux du modèle.



Après une rapide recherche, nous avons pu régler ce problème. Il suffisait de changer le shader standard vers le shader Universal Render Pipeline.



Finalement, les textures initialement corrompues ont pu être corrigées



2.6 Système d'HP / mort et de combat sur des Game Object

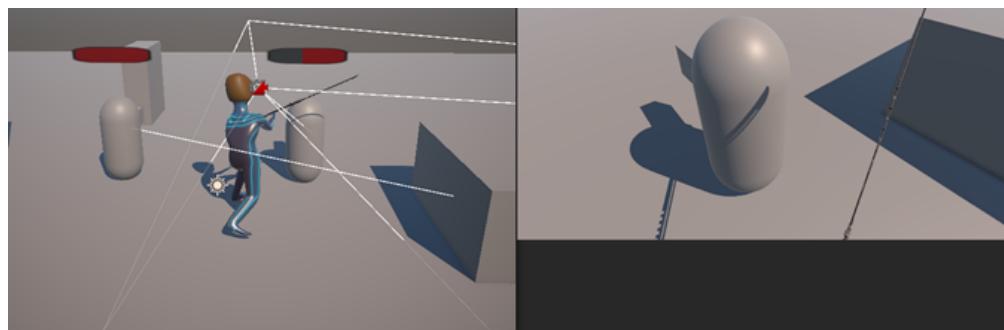
Tâche réalisée par Laurent ZHOU

L'interaction, autrement dit, le système de combat, de vie et de mort, est gérée par la classe logique. L'interaction est principalement possible à l'aide de raycasts qui existent dans Unity. Un raycast est un rayon invisible qui a un point de départ, une longueur et une direction défini. Ce rayon peut retourner "true" s'il détecte un objet et "false" sinon. Par ailleurs, il existe aussi une fonction qui permet de récupérer l'objet que le rayon a détecté.

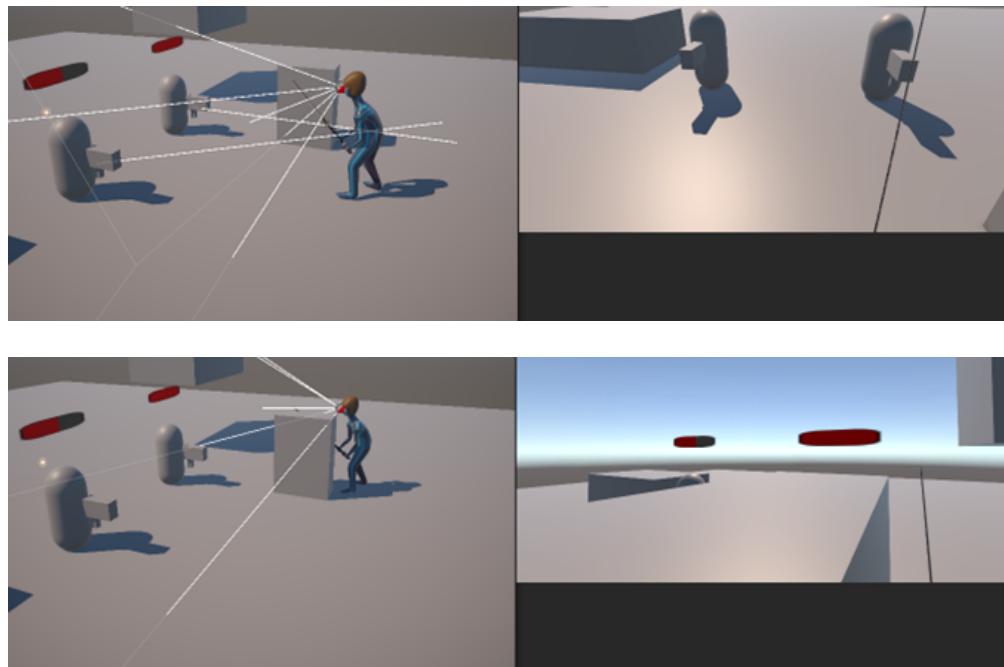
De plus, Il faut également créer l'objet "Enemy" qui représente une entité hostile. Dans l'objet, on ajoute une propriété privée de type int qui représente ses PV. Ainsi, une méthode qui permet d'enlever ses PV est ajoutée, et prend en paramètre le nombre de dégâts infligés. Dans l'objet PC, on commence par déclarer deux propriétés privées, une de type "Enemy" qui représente l'ennemi ciblé et une autre "Damage" de type int qui représente les dégâts que le joueur (PlayerController) peut infliger à l'ennemi en un coup. Par la suite, on déclare

une méthode privée qui met à jour chaque frame de l'ennemie couramment sélectionnée à l'aide du raycast, si aucun ennemi n'est sélectionné, la propriété devient nulle.

Pour finir, on ajoute une nouvelle méthode qui appelle la fonction de l'ennemi sélectionné permettant de lui infliger des dégâts.



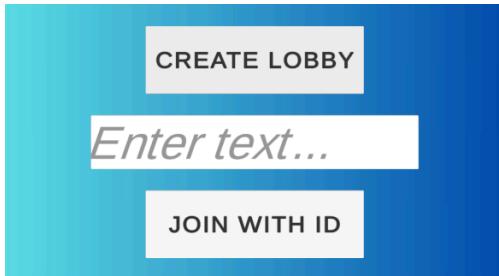
L'ennemi doit également être capable de riposter. Pour se faire il doit, dans un premier temps, déterminer le joueur le plus proche puis vérifier si il se trouve dans son champ d'action grâce au raycast.



2.7 Menu

Tâche réalisée par Miles HOLLAND

Pour offrir une expérience complète, nous avons créé un canvas permettant d'avoir une interface d'utilisateur conviviale pour la connexion. Bien que temporaire, la création du menu permet aux joueurs de naviguer facilement dans le processus de connexion et de rejoindre rapidement des parties multijoueurs sans difficultés.



2.8 Problèmes / Solutions à envisager

Nous serons sûrement confrontées à des problèmes de collisions pour les ennemis, ce qui n'est pas le cas pour l'instant grâce à l'utilisation du Package NavMesh. De plus, il pourrait y avoir un problème lorsque les joueurs se retrouvent face au même ennemi. Quel sera le comportement de celui-ci ? Par ailleurs, il pourrait y avoir des bugs lors des animations des ennemis ou du personnage. Toutes ces situations sont à envisager avant la soutenance finale pour mener à bien notre projet. Les solutions envisagées seraient dans un premier temps, de rentrer les BoxColisions des ennemis plus grandes pour éviter des problèmes de collisions entre les joueurs et les ennemis. Concernant les ennemis et le multijoueur, il faudrait faire un script qui calcul la distance entre l'ennemi et le joueur le plus proche. Ainsi l'ennemi n'aura que d'autres choix que d'attaquer le premier joueur qui se trouve dans son périmètre, et il ne pourra pas changer de cible. Enfin, il faudra bien faire attention à assigner les bonnes actions dans l'animateur de l'ennemi et du player afin qu'il n'y ait pas de problèmes entre le script et l'animateur. Il suffit d'une erreur dans le script pour que l'ennemi fasse la mauvaise action et rende le jeu inutilisable.

3 TÂCHE À RÉALISER LORS DE LA SOUTENANCE FINALE

3.1 Créer un système d'XP/Gold

Il faudra créer des attributs expérience (XP) et Gold (Or). Lorsque l'ennemi sera tué, il suffira d'ajouter des points aux attributs XP et Gold. Nous pourrions par exemple, ajouter une barre de vie à nos Player via des Canvas, c'est-à-dire implémenter une barre de rouge sur l'écran du joueur qui représente sa vie (de même pour l'argent que recevra le joueur (Gold)).



Exemple de barre d'XP et d'HP

3.2 Amélioration des Assets

Les Assets / Artworks du level design, ennemi ou du joueur peuvent être amené à changer. En effet, lors de cette première soutenance, notre but est de montrer que les principales fonctionnalités de notre jeu fonctionnent. Les Assets n'étant pas la priorité, nous pourrions changer de DA lors de la soutenance finale.

3.3 Musique

Les musiques permettent d'immerger le joueur au sein de notre jeu. Nous avons déjà commencé à implémenter des musiques que ce soit pour le menu, les boss et même dans les niveaux. Nous avons pris de l'avance concernant cette partie.

3.4 Level design

Le Level Design consiste à conceptualiser et élaborer toutes les facettes d'un niveau au sein du jeu. Cette partie va nous prendre du temps à réaliser puisqu'il faudra imaginer des niveaux dans lequel les joueurs évolueront. De plus, Nous avons déjà réalisé un monde SandBox, c'est-à-dire, un environnement qui regroupe toutes les fonctionnalités primordiales de notre jeu, un multijoueur fonctionnel, un personnage qui peut marcher, courir, sauter, s'accroupir, Wallride et Walljump sur les murs, des ennemis corps-à-corps et à distance et un système de combat.

4 CONCLUSION

Nous avons effectué les fonctionnalités primordiales pour réaliser un jeu vidéo, à savoir un multijoueur, des ennemis (corps-à-corps et à distance), des déplacements, un système de vie, de mort, un menu, des objets de terrains et des assets (toujours en cours). Nous avons créé un monde SandBox avec toutes ces fonctionnalités, afin de pouvoir tester différents problèmes qui pourrait arriver. Nous avons donc respecté notre cahier des charges jusque-là. Ainsi, Nous pouvons continuer la conception de notre jeu en implémentant les différentes fonctionnalités qui vont nous permettre de créer un jeu vidéo fonctionnel.

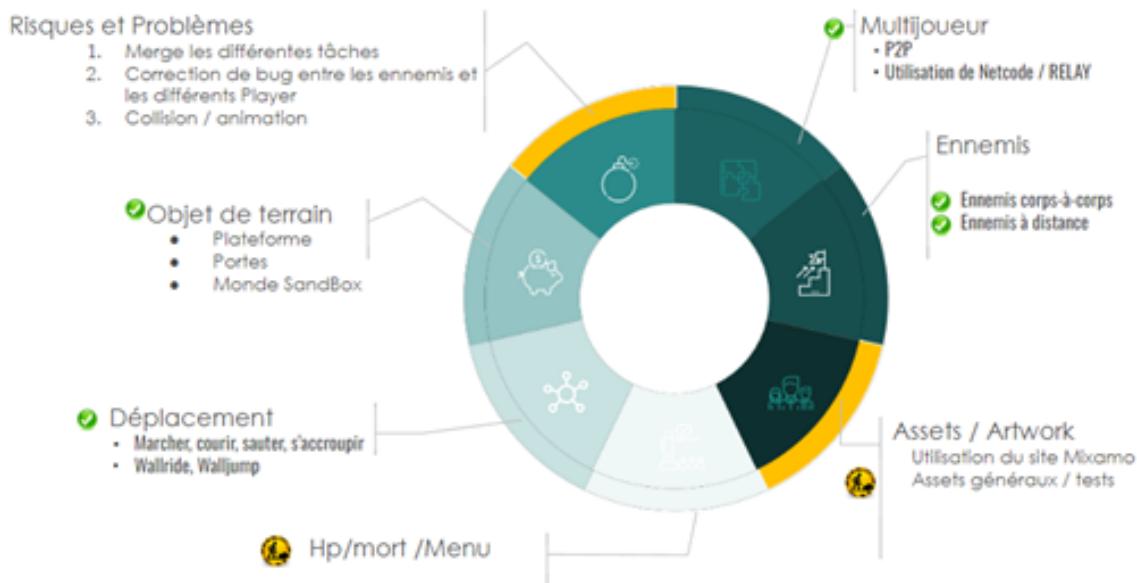


Diagramme circulaire des fonctionnalités après première soutenance