


Rapport de soutenance 1

Jason, Rémi et Arthur

7 Mars 2023

image/LOGO.png

Contents

I	Introduction	4
II	Bilan de Rémi	5
1	Les Avancées	5
1.1	Le Mage	5
1.2	Le site internet	6
2	Les nouveautés	9
2.1	L'assassin	9
2.2	Le menu de fin de jeu	10
3	Conclusion de Rémi	11
3.1	état actuel	11
3.2	Reste a faire	11
4	Site Internet	12
4.1	Introduction	12
4.2	Etat actuel	12
4.3	Reste a faire	14
5	Systeme d'amélioration	14
5.1	Les débuts	14
5.2	Mon avancement	14
III	Bilan de Jason	14
6	Les ennemis	15
6.1	Enemy Movements	15
6.2	Enemy Animations	17
7	Les maps	17
8	Problèmes rencontrés et retards	19
IV	Bilan de Arthur	19
9	Le guerrier	20
9.1	Attack()	20
9.2	Skill()	21

10 Multijoueurs	21
10.1 Connection des joueurs sur un même serveur	21
10.2 création d'un personnage par joueurs	22
10.3 Menu de connection	23
11 L'archer	23
11.1 Attack()	24
11.2 Skill()	24
12 Mes impressions	24

Part I

Introduction

Nous avons globalement une bonne base du jeu. En effet, nous avons une carte, le déplacement du joueur, des ennemis équipés d'un système de détection des joueurs et qui se déplacent automatiquement vers les joueurs. Il y a aussi différents personnages implémentés (certains plus complets que d'autres...). Nous avons également le système d'interaction entre le joueur et le Zombie, et enfin un système de Multijoueurs. Nous avons donc un bon morceau du jeu, il reste encore cependant des ajouts/perfectionnements à implémenter. Pour la suite

du Rapport, chacun expliquera ses tâches effectuées, comment il a réussi à les effectuer, les problèmes qu'il a rencontré et ce qu'il lui reste à faire.

Part II

Bilan de Rémi

1 Les Avancées

1.1 Le Mage

La dernière fois, j'avais seulement commencé à développer le personnage "Mage". Cette fois-ci, j'ai quasiment terminé son implémentation et il est parfaitement fonctionnel. Pour rappel, le Mage est l'un des différents types de personnages jouables par le joueur. Il est équipé d'un bâton magique et peut tirer de loin, mais ne fait pas beaucoup de dégâts. Il a cependant une compétence qui lui permet de soigner ses alliés et de repousser les monstres.

Pour la première soutenance, j'avais commencé à implémenter le système de tir, mais celui-ci bien que fonctionnel, ne collait pas tout à fait au jeu. En effet, on pouvait tirer sans limite de temps, c'est-à-dire que l'on pouvait tirer autant de fois que l'on voulait par seconde. Cela était beaucoup trop fort et rendait le jeu injouable et trop facile. Pour remédier à ce problème, j'ai donc implémenté un système de chronomètre qui fait que le Mage ne peut tirer qu'une fois toutes les 1 seconde, soit au maximum 1 tir par seconde. Ce qui rend le jeu plus réaliste, moins chargé graphiquement et plus jouable. Pour faire ce système de chronomètre, j'ai utilisé la fonction "Time.deltaTime", que j'ai d'ailleurs trouvé facile d'utilisation et assez pratique.

J'ai également profité de l'ajout de ce chronomètre pour régler un petit souci au niveau de la balle tirée. En effet, celle-ci ne s'arrêtait jamais à moins de rencontrer un objet. J'ai donc remédié à ce problème en détruisant la balle au bout de 2 secondes si elle n'avait rien rencontré. Finalement, ces petites modifications ne m'ont pas pris beaucoup de temps et m'ont surtout permis de me familiariser avec le système de temps dans Unity. De plus, j'ai veillé à ce que la balle ne puisse infliger des dégâts qu'aux zombies et non aux joueurs. En effet, en multijoueur, cela aurait pu être un peu gênant car les joueurs auraient pu s'infliger des dégâts entre eux.

Je me suis ensuite attaqué à l'autre atout du Mage : sa capacité à soigner ses alliés. Pour cela, j'ai créé une sorte d'aura autour de lui qui est remarquable par son halo jaune :

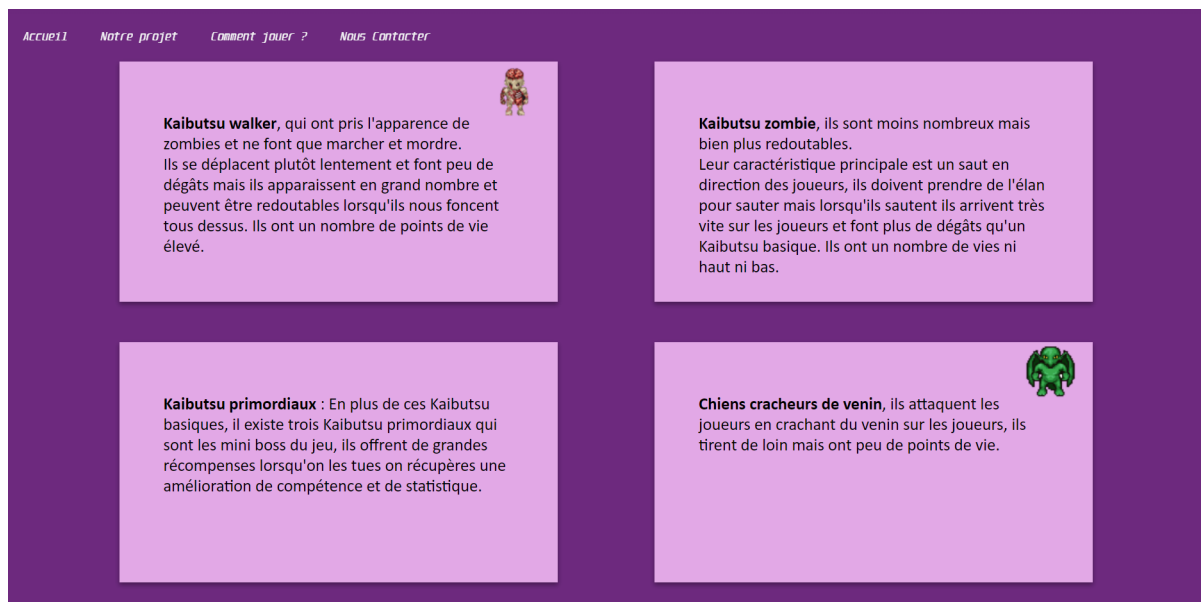


Cette capacité ne se déploie que lorsque le joueur appuie sur la touche "e" et est accompagnée, là aussi, d'un chronomètre de telle sorte que la capacité ne dure que 6 secondes et ne peut s'activer que toutes les 20 secondes. Le système de soin est le suivant : toutes les secondes, chaque joueur se trouvant à l'intérieur du cercle jaune gagne 2 points de vie. Cet ajout de capacité a été un peu difficile techniquement et m'a pris un peu plus de temps que prévu. En effet, la partie la plus difficile était de faire bouger le cercle et le joueur en même temps, car au début, le cercle n'apparaissait qu'à l'endroit où le personnage était au moment où le joueur a appuyé sur "e". Pour remédier à ce problème, j'ai donc dû donner des caractéristiques de déplacement au cercle jaune pour qu'il puisse suivre le joueur et rester tout le temps au centre de lui.

1.2 Le site internet

La dernière fois, j'avais créé un site internet avec 4 pages : Accueil/Comment jouer/Notre projet/Nous contacter. J'avais exposé brièvement le concept du jeu, fait une petite présentation de notre groupe ainsi qu'une explication des personnages et des touches de jeu. Aujourd'hui, j'ai donc ajouté plusieurs éléments permettant de compléter la présentation du jeu.

Tout d'abords j'ai ajouté une description des ennemies :

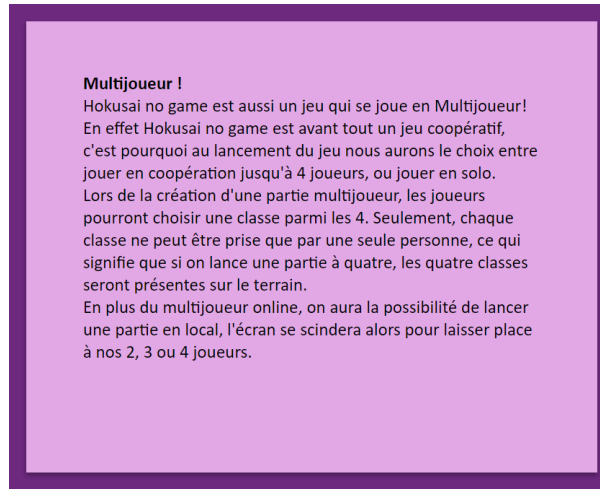


J'ai ensuite ajouté les liens de nos sources comme demandé, c'est-à-dire tous les liens (des membres, des logiciels, des images, des sons, des bibliothèques, des applets et autres éléments que nous avons utilisés) :

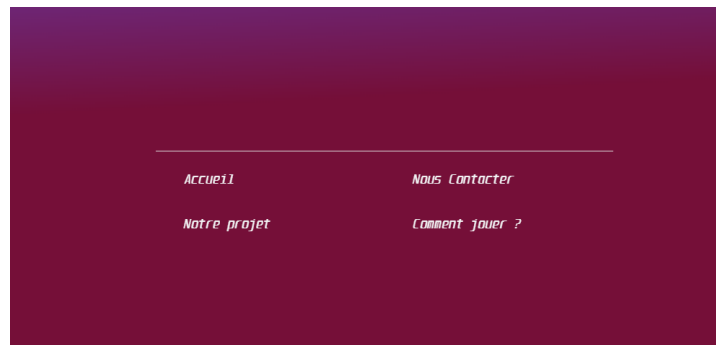


Puis j'ai ensuite rajout quelques petites information et mise en forme :

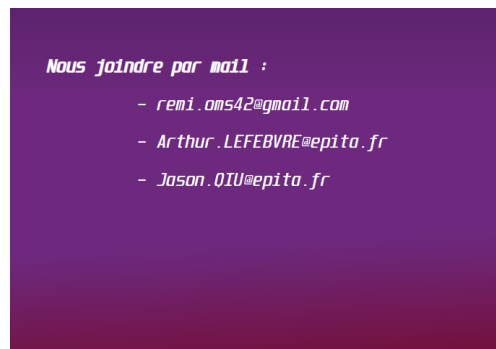
Dans la fin de la presentation du jeu :



En bas de chaque page pour naviguer plus facilement



J'ai également ajouté nos mails pour nous contacter plus facilement car le formulaire automatique n'est pas supporté dans tous les moteurs de recherche.



2 Les nouveautés

2.1 L'assassin

Parmi les nouveautés que j'ai implémentées, il y a l'Assassin. C'est également l'un des personnages jouables par le joueur. Pour rappel, il est équipé d'une dague et peut infliger de très gros dégâts seulement lorsqu'il est proche des ennemis. De plus, il peut devenir invisible pendant un court instant. Enfin, il a aussi une meilleure vitesse de déplacement que les autres personnages.

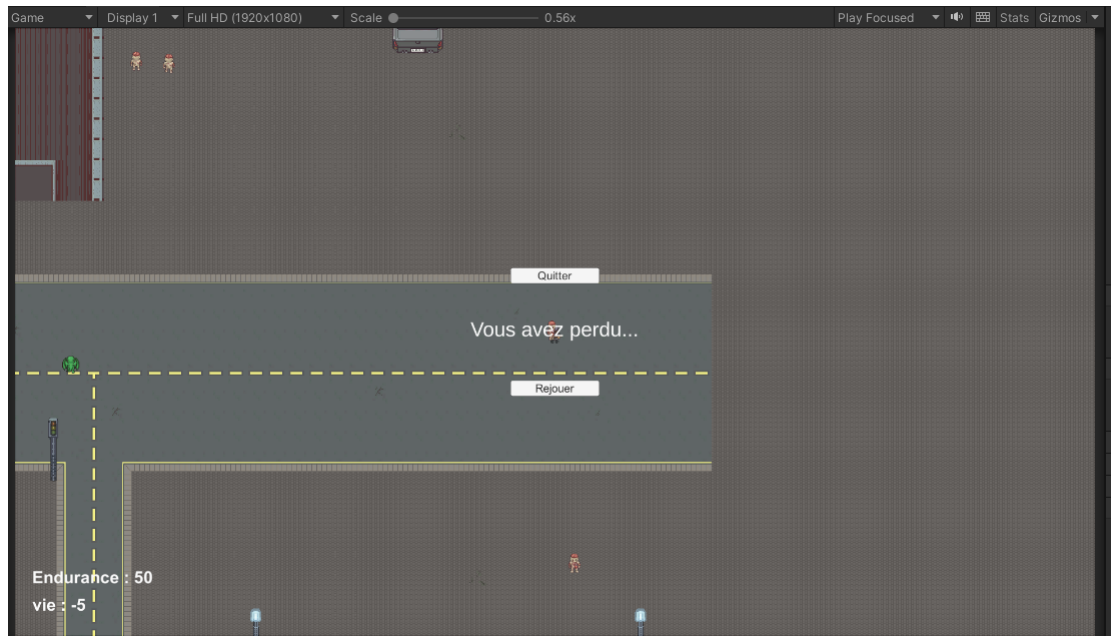
Pour commencer l'implémentation de ce personnage, je me suis inspiré de ce qui avait déjà été fait avant pour le mage et le guerrier afin de gagner du temps. Au niveau de l'attaque, il a presque la même que le guerrier, c'est-à-dire des coups d'épée au corps à corps. Cela n'a donc pas été trop difficile à faire, il y a juste une petite adaptation au niveau des dégâts. De plus, il y a aussi une petite adaptation de la vitesse de déplacement et des points de vie. Au niveau de l'apparence graphique, nous lui avons donné pour l'instant l'apparence du mage, car nous n'avons pas encore trouvé une apparence qui lui convient, mais nous lui en donnerons une qui lui permettra d'être différencié du mage.

Le vrai défi dans l'implémentation de l'Assassin a été sa capacité. En effet, comme rappelé plus haut, il peut devenir invisible pendant un court instant. Au début, je ne savais pas trop comment m'y prendre ni par où commencer. J'ai essayé de le faire disparaître, mais il ne réapparaissait pas. J'ai ensuite essayé de le cacher derrière un objet, mais les zombies le suivaient toujours. C'est alors que j'ai eu une idée : j'ai changé le "tag" de l'Assassin de "player" à celui des zombies "enemies". De ce fait, le personnage reste visible sur la carte (pour rendre le jeu plus facile pour le joueur), mais l'Assassin est alors "invisible" pour les ennemis. En effet, ceux-ci le considèrent alors comme l'un des leurs et ne l'attaquent plus ni n'avancent vers lui. Mais attention, car là aussi cette capacité n'est pas infinie. Elle ne dure que 10 secondes.

Cette capacité a été intéressante à implémenter et je trouve qu'elle est assez agréable à jouer car elle permet de s'infiltrer parmi les zombies. Cependant, l'on pourrait alors penser que l'Assassin subit des dégâts de la part de ses alliés quand il "devient un zombie", mais non, il n'en prend pas. J'aurais pu faire en sorte qu'il en prenne, mais je me suis dit que cela risquerait de rendre le jeu plus compliqué en équipe.

2.2 Le menu de fin de jeu

L'une des autres nouveautés que j'ai implémentées est le menu de fin de jeu. Celui-ci apparaît, comme son nom l'indique, à la fin du jeu lorsque le joueur meurt. Il est composé de deux boutons : l'un pour quitter le jeu et l'autre pour recommencer une partie :



L'implémentation de ce menu de fin n'a pas été très facile et m'a pris pas mal de temps. En effet, j'ai dû trouver comment l'afficher, puis comment faire des boutons fonctionnels.

Pour l'afficher, j'ai créé un Canvas dans lequel j'ai créé un Panel contenant le texte et les boutons. Ensuite, j'ai relié ce Canvas au script des personnages de telle sorte qu'il apparaisse et stoppe le jeu lorsque le joueur a moins de 0 point de vie.

Puis pour les boutons, j'ai essayé de nombreuses options différentes mais qui ne marchaient pas. J'ai donc finalement réussi à faire un bouton "Rejouer" qui, lorsqu'il est cliqué, ramène au menu de début. Et j'ai aussi réussi à faire le bouton "Quitter" qui, lorsqu'il est cliqué, quitte le jeu. En somme, j'ai donc réussi à faire un menu de fin fonctionnel, même s'il peut encore être amélioré en rajoutant par exemple plus de boutons et en le rendant plus beau.

3 Conclusion de Rémi

3.1 état actuel

J'ai donc réussi à finir ce que j'avais dit que je finirais la dernière fois, c'est-à-dire principalement le mage, l'assassin et le site internet. Et j'ai aussi réussi à implémenter quelques nouvelles choses, comme le menu de fin. À noter que j'ai aussi corrigé quelques petits dysfonctionnements par ci par là, ce qui m'a pris au final un peu de temps tout de même.

3.2 Reste a faire

Pour la fin du projet, je compte faire tout ce qui est relié au système d'amélioration, ce qui représente une grosse tâche tout de même. Ensuite, je ferai quelques petits ajouts pour rendre le jeu encore plus interactif, comme par exemple des bulles de dialogue ou encore des Easter eggs.

J'ai commencé par le système de tirs. Cela n'était pas facile. En effet, j'ai dû créer une classe, puis une autre pour les balles afin de mieux gérer leur comportement. Et ensuite il a fallu la relier à une image (la balle qui est tirée). Le plus compliqué a été de faire en sorte que la balle tirée parte en direction de là où est le pointeur de la souris. Une fois cela fait, j'ai géré le cas où la balle rencontre un quelconque objet et à ce moment là disparaît. Ou alors, si elle ne rencontre rien, disparaît au bout d'un certain moment. Cette fonctionnalité a été créée afin que le Mage ne puisse pas tirer à l'autre bout de la carte.

Il me reste encore à gérer le cas où la balle rencontre un zombie, afin de pouvoir lui enlever des points de vie, tout comme les autres caractéristiques du Mage, tel que le soin des alliés. Je pense néanmoins que ces ajouts me prendront certes du temps, mais seront nettement plus faciles, dû à ma prise en main de l'outil.

4 Site Internet

4.1 Introduction

Pour le site internet, je savais que cela n'allait pas me poser de soucis au niveau de mes compétences techniques. En effet, j'ai déjà créé un site internet pour une entreprise. Je me suis donc naturellement mis sur la création du site internet. J'ai créé le site en utilisant principalement le langage HTML / CSS, ainsi qu'un peu de PHP pour le côté interactif. À noter que son développement est encore en cours et ne sera complètement fini que lorsque le jeu lui-même ne sera terminé. J'ai quand même créé une grande partie de la base du site.

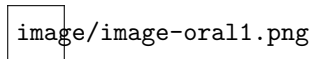
4.2 Etat actuel

Le site a actuellement 4 pages: la principale (Accueil) où je présente le jeu et son histoire avec des photos pour illustrer les propos. Viens ensuite la page "Notre Projet" où j'explique, comme son nom l'indique le déroulement de notre projet, c'est à dire comment nous avons eu l'idée du jeu, notre vision de celui-ci, les différentes tâches et touches originales... Nous avons ensuite la page "Comment jouer" dans laquelle j'explique comment jouer : les touches, les astuces, le but du jeu, les niveaux... Et enfin il y a la page "Nous contacter" dans laquelle j'ai mis un formulaire à remplir et qui est directement envoyé à mon adresse mail.

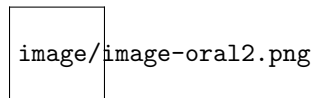
J'ai commencé par créer la barre de menu, celle qui nous permet de naviguer entre les différentes pages. J'ai ensuite cherché la bonne mise en forme : choisir les bonnes couleurs, l'agencement des textes, si je mettais des images du jeu, des animations... Ce travail m'a pris un peu de temps de réflexion mais je me suis principalement laissé guider par mon inspiration au fur et à mesure de l'avancée du site.

Pour la premiere page, j'ai eu l'idée de faire un dégradé progressif de couleur entre le noir (en haut de page) et le violet (en milieu de page) avec le nom du jeu en grand au début. Ceci permet de donner une première image du site assez originale qui nous plonge dans l'univers du jeu.

Image du debut du site :



Ensuite j'ai ajouter le texte, sous forme de petit bloc carré, afin d'apporté de la légèreté au texte et de le rendre plus en adéquation avec le style du jeu.
Image d'une partie de texte :



J'ai ensuite, et cela pour toutes les pages, mis une forme traditionnelle de fin de page, dans laquelle on peut nous contacter, ou tout simplement trouver des informations supplémentaires :

Pour les 2eme et 3eme, j'ai simplement repris le code de la première en le réadaptant, c'est à dire modifier le texte, et les images. C'est pour la dernière page que les choses se compliquent. En effet, je voulais faire un espace où l'on pouvait écrire un message qui me serait directement envoyé par mail. En notant que le message ne peut être envoyé si le formulaire est incomplet. J'ai donc réussi à faire cette zone de message mais le problème est dans l'envoi du mail, cela ne fonctionne pas encore.

Image de la zone de message :



image/image-oral3.png

4.3 Reste a faire

Il me reste encore du texte/image à rajouter au fur et à mesure que le jeu avance. Concernant le texte, se sera principalement les différent(e)s étapes/problèmes rencontré(e)s et les solutions apportées pour y remédier. Il me restera aussi à régler le problème de l'envoi du message. Ainsi que un lien permettant de télécharger le jeu, mais cette fonctionnalité sera faite en dernière. Il faudra aussi que je crée une nouvelle page "Nos ressources" dans laquelle je mettrai des liens envoyant vers les différents sites utilisés pour nos images de jeu/ logiciel/ et autres éléments externes.

5 Systeme d'amélioration

5.1 Les débuts

J'ai commencé les améliorations par la potion lâchée par les zombies à leur mort. Pour cela, j'ai simplement créé une nouvelle classe "Potion" qui correspond à l'objet potion. Puis, j'ai fait en sorte que quand le zombie meurt, il instancie une potion.

Cette potion reste là où le zombie est mort et ne disparaît que lorsqu'elle est touchée par un autre objet (zombie, joueur, balle). Il me reste à faire en sorte que quand un joueur touche la potion il gagne des points de vie. Concernant les autres systèmes d'améliorations, celles-ci seront implémentées au fur et à mesure du jeu. Mais, connaissant maintenant comment gérer ces systèmes, leur implémentation sera plus facile et prendra moins de temps à l'avenir.

5.2 Mon avancement

Bien que je n'ai pas implanter beaucoup de choses (système d'amélioration, personnages, écran de pause, de début et de fin) et que mes camarades ont, sur le jeu, effectués plus de choses que moi, je reste néanmoins confiant dans mon avancée, et vis à vis du planning. Même si j'ai pris du retard sur certaines tâches (dû à la perte d'un membre mais aussi en raison de la prise en main de Unity et de sa compréhension, ceci a été plus difficile que je ne le pensais). J'ai préféré bien faire les tâches qui me semblaient les plus intuitives (site internet) même si elles m'ont prises pas mal de temps, ce qui peut expliquer la différence de système implémentée dans le jeu.

Part III

Bilan de Jason

6 Les ennemis

6.1 Enemy Movements

J'ai commencé le projet par l'implémentation des déplacements du Kaibutsu Walker, c'est-à-dire l'ennemi de base. J'ai fait en sorte qu'il se déplace de manière aléatoire lorsqu'il n'y a aucun joueur à proximité mais lorsqu'un joueur se rapproche suffisamment, le zombie commence alors à le pourchasser.

Pour le système de déplacement aléatoire du Walker, j'ai utilisé un système de waypoints.

Lorsque le zombie est en mode déplacement aléatoire, un waypoint, point de destination, sera alors créé à une distance plus ou moins proche de ce dernier de manière aléatoire, et l'ennemi se rendra à cette destination ; après l'avoir atteint, un nouveau waypoint sera créé et ainsi de suite jusqu'à que le zombie meurt ou qu'il rencontre un joueur à poursuivre.

Pour le système de poursuite du joueur, je récupère la position du joueur, je calcule la direction dans laquelle le zombie doit se diriger puis je fais déplacer le zombie vers cette direction. Lors de la poursuite du joueur, un nouveau waypoint est créé à chaque frame car comme je l'ai dit précédemment, un waypoint est créé uniquement après que l'ennemi s'y soit rendu, or lorsque qu'un ennemi commence à pourchasser un joueur le waypoint qui été créé n'est pas actualisé ce qui fait que lorsque l'ennemi arrête sa poursuite il va vouloir aller au dernier waypoint créé avant sa poursuite: résultat, il peut se retrouver à faire une très longue ligne droite pour retourner à ce waypoint, donc pour éviter cela et pour rendre son déplacement plus réaliste, j'actualise le waypoint à chaque frame lors de la poursuite.

Par la suite, j'ai amélioré le système de poursuite du zombie pour l'adapter au multijoueur. Pour la première version de ChasePlayer(), la méthode qui permet à un ennemi de poursuivre un joueur à proximité, j'utilisais le GameObject Player qui permettait de connaître la distance du joueur à tout instant et donc de savoir quand est ce qu'il fallait que le zombie commence à poursuivre le joueur. Cependant cette méthode ne pouvait pas fonctionner quand il y avait plusieurs joueurs à proximité car cela faisait buguer le zombie ne sachant pas qui poursuivre entre les 2 joueurs.

Il a fallu donc changer cela: à la place détecter si un joueur est à proximité d'un ennemi, j'utilise la méthode OverlapCircleAll() de la classe Physics2D de Unity, qui permet de récupérer le Collider2D de tous les objets se trouvant

dans le cercle de rayon R du zombie. Je vais en même temps vérifier si l'objet est bien un joueur et s'il est bien visible par le zombie. Un joueur est visible par un ennemi si entre l'ennemi et le joueur il n'y a aucun mur, bâtiment ou obstacle trop gros et bien évidemment . Pour faire cela, je vais utiliser la méthode `Linecast()` de la classe `Physics2D`, qui trace un trait entre 2 points: ici ça sera notre objet et l'ennemi, le trait s'arrête à la première cible ayant le mask "Action". Tous les obstacles volumineux, bâtiments et murs du jeu ainsi que les joueurs ont le mask "Action". assez volumineux) se trouve entre le joueur et l'ennemi ou si le joueur est trop loin de l'ennemi c'est à dire qu'il n'est pas dans le cercle de détection, le joueur ne sera pas considéré comme un visible par l'ennemi . Si un joueur est visible, il sera ajouté au tableau Le trait va récupérer le `Collider` de la première cible touchée, grâce à cela on peut comparer le tag de l'objet et voir si son tag est "Player". A partir de cette méthode, si un obstacle(`visibleTargets` qui contiendra tous les `Collider2D` des joueurs visible par un ennemi. L'inconvénient de cette méthode est qu'elle va récupérer les objets non joueurs et les tester pour savoir si ce sont des joueurs.

Cette méthode `CanSeePlayers()` sera appelée dans la méthode `UpdateMovement` , ce qui permettra de savoir à chaque frame les joueurs visibles par chaque ennemi : elle renverra `True` s'il y a au moins un joueur visible par l'ennemi et `False` sinon. Si un seul joueur est visible par un ennemi, alors il aura juste à poursuivre de dernier mais si plusieurs joueurs sont visibles alors il faudra déterminer lequel est le plus proche. Pour cela, on parcourt le tableau de tous les joueurs visibles `visibleTargets[]` qui a une taille de 4 car le nombre maximum de joueurs pouvant jouer en même temps est de 4, puis on détermine le joueur le plus proche de l'ennemi qui sera le `NearestPlayer()`.

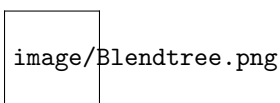
Je combine le tout dans `UpdateMovements()`: je vérifie si l'ennemi peut voir au moins un joueur, si oui il poursuit le joueur le plus proche, sinon il déplacera de manière aléatoire comme un zombie.

Il y a un autre détail qui m'a posé quelques problèmes, pour que le jeu soit réaliste il faut que les ennemis ne puissent pas traverser les obstacles et bâtiments. Je lui ai donc rajouté un `collider2D` ainsi qu'un `rigidbody2D`. J'ai enlevé toute physique dans le `rigidbody` pour ne pas que l'ennemi tombe dans le vide. J'ai verrouillé l'axe Z pour que le zombie ne puisse pas tourner. Un autre problème est survenu : lorsque le joueur le poussait , il s'envolait à l'infini dans la direction dans laquelle le joueur l'avait poussé ce qui était un gros problème car ça cassait tout le réalisme. J'ai du réfléchir à une solution pour résoudre cela, j'ai essayé de verrouiller l'axe X et Y ce qui faisait que le joueur ne pouvait plus du tout le pousser, ce n'était pas non plus ce que je voulais, en plus de cela, en faisant cela le zombie pouvait traverser les murs pour aucune raison. A la base pour déplacer un ennemi, je me servais du `Component Transform` mais en déplaçant son `rigidbody` plutôt que son `Transform`, tout a été résolu. Le joueur pouvait pousser le zombie mais ce dernier ne s'envolait pas à l'infini. J'ai utilisé la méthode `MovePosition` de la classe `Rigidbody2D` qui permet de déplacer un `rigidbody2D` selon un vecteur donné.

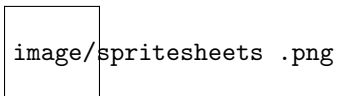
6.2 Enemy Animations

Pour l'animation de l'ennemi, j'ai utilisé une fonctionnalité de Unity appelée Blend Tree, grâce à cet outil j'ai pu implémenter les animations de marche du Kaibutsu Walker (l'ennemi de base) sans utiliser une quantité astronomique de fonctions. Un ennemi possède 4 animations de marche : vers le haut, vers le bas, vers la gauche, vers la droite. Faire en sorte de mettre la bonne animation selon le déplacement de l'ennemi est vraiment compliqué avec seulement du code, mais avec l'outil Blend Tree de Unity, on peut facilement attribuer une animation avec un déplacement du zombie.

Je vais expliquer plus en détail comment j'ai fait: Tout d'abord pour bien coordonner les animations avec le déplacement du zombie, il faut connaître la direction dans laquelle il marche à chaque frame. On va donc la calculer en soustrayant sa position actuelle avec son ancienne position qui seront calculées aussi à chaque frame, ce qui va nous donner un Vector2 direction. On va créer 2 float X et Y qui seront les composantes qui vont nous servir à changer les animations de marche quand il le faudra; elles sont comprises entre -1 et 1. La composante X est la direction horizontale et Y la composante verticale, ensuite il ne reste plus qu'à mettre les bonnes valeurs de X et Y sur le Blend Tree puis à mettre les animations correspondantes à chaque direction. Unity va lui-même se charger d'afficher la bonne animation selon les composantes X et Y.



Les spritesheets du Kaibutsu Walker ont été créés et générés à partir d'un site de création de spritesheets pour personnages humanoïdes de type pixel art 2D. Cela m'a pris beaucoup de temps de trouver un site qui permet de créer des spritesheets aussi facilement et rapidement.

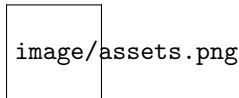


Pour créer les animations, j'ai importé tous les spritesheets des animations, ajouté chaque animation au Blend Tree. Pour chaque animation, j'ai sélectionné les PNG constituant l'animation puis j'ai ajusté les frames d'animation pour rendre le tout plus fluide et jolie.

7 Les maps

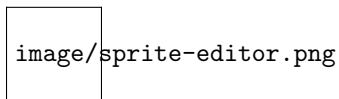
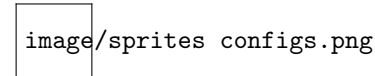
Je me suis aussi chargé de créer les maps du jeu. Comme notre jeu est de type pixel art 2D, j'ai choisi d'utiliser le système de tilemap, qui permet de créer

des maps pixel art de manière simple et rapide(même si j'ai pris beaucoup de temps à comprendre comment ça marchait) Notre jeu se passe dans une ville post-apocalyptique, l'ambiance doit être assez pesante et les assets utilisés doivent être dans le thème. J'ai passé des heures à chercher un asset de ville qui conviendrait à l'ambiance du jeu mais avec pas trop de relief pour ne pas compliquer trop la création des maps. J'ai alors opté pour cet asset gratuit trouvé sur le site OpenGameArt.

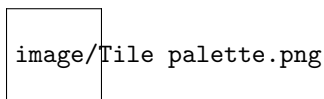


Ensuite, après avoir téléchargé l'asset il faut configurer les Tiles pour pouvoir les utiliser. Lorsqu'on télécharge l'assets, on se retrouve avec seulement des images PNG, on va alors les traiter pour les transformer en Tiles et les utiliser.

Tout d'abord j'ai mis le sprite mode en multiple car sur une image PNG on avait différentes tiles, le pixels per unit à 32 car les tiles faisaient du 32x32 pixels, le filter mode en "point" adapté pour le pixel art. Ensuite, il faut séparer les tiles en ouvrant le sprite editor, aller dans la section slice et séparer les tiles en mettant Type à "Grid by cell size" et Pixel Size à 32.



Après avoir fait cela sur toutes les tiles à utiliser, on peut enfin créer les map mais avant ca il faut créer une palette. Comme en peinture , cela va nous servir à dessiner la map en choisissant les tiles qu'on veut utiliser. Grâce à l'outil palette, créer une map est beaucoup plus simple car on peut littéralement faire de la peinture avec les tiles.

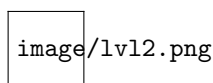
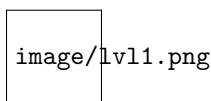


La palette a été créée, maintenant tout ce qu'il reste à faire est de créer la map. Sauf que pendant la création du niveau, j'ai rencontré un problème celui de superposer une tile sur une autre pour par exemple ajouter des éléments décor tels que des voitures ou encore des barrières. Pour remédier à cela, j'ai

dû créer plusieurs tilemap, chaque tilemap possède un niveau de Layer qui est l'ordre hiérarchique de superposition: une tilemap avec un layer de 7 sera au-dessus d'une tilemap avec un layer de 6. Grâce à cette mécanique, on peut superposer des tiles sur d'autres et l'ensemble de ces tiles vont former la map globale : la grille.

Après avoir terminé la création d'un niveau, il faut rajouter des Colliders aux bâtiments, barrières, voitures etc pour que les joueurs et les ennemis ne puissent pas les traverser. Pour cela, j'ai ajouté un component appelé Tilemap Collider 2D qui permet de rajouter un Collider 2D sur tous les éléments d'une tilemap puis j'ai aussi rajouté un Composite Collider 2D pour les éléments avec un gros collider ce qui permet d'avoir un gros collider global pour chaque objet et donc une meilleure optimisation du jeu au lieu de pleins de plusieurs petit colliders.

J'ai pu réalisé 2 maps: le premier est un endroit de la ville avec un grand carrefour au centre et quelques bâtiments et le deuxième est un grand parking abandonné.



8 Problèmes rencontrés et retards

Quelques jours avant la soutenance, je me suis rendu compte que l'animation du Kaibutsu Walker (le zombie) était bugué, l'animation de marche saccadait, on voyait que l'animation marchait mais qu'elle changeait entre chaque frame avec l'animation par défaut ce qui faisait qu'on avait l'impression qu'il buguait. Pourtant l'animation marchait bel et bien quand je l'avais implémentée pour la première fois mais après avoir reclone le projet, c'est là que le problème est apparu. J'ai du donc changer la méthode de mouvement du zombie et repasser avec la méthode en utilisant le component Transform mais en contre partie, un autre problème est survenu, le tout premier, celui que lorsqu'un joueur pousse un ennemi, ce derniers s'envole a l'infini.

Je n'ai pas pu finir toutes les animations et interactions du zombie car j'étais aussi beaucoup occupé avec la création des cartes.

Part IV

Bilan de Arthur

9 Le guerrier

Pour le début du projet j'ai commencé par créer un personnage simple à l'apparence d'un cube qui est capable de faire des mouvements basiques dans l'espace. Le personnage est capable de se déplacer dans les 4 directions ainsi que de courir en appuyant sur shift. Pour se faire je récupère grâce à la méthode `Input.GetAxis` la direction dans laquelle veut se déplacer le joueur puis je crée un vecteur de vitesse du joueur en multipliant l'axe avec une vitesse prédéfinie dans une variable privée `speed`. Je modifie cette variable lorsque le joueur appuie sur shift et je rajoute un booléen qui indique si le joueur est entrain de courir ou pas et lorsque le joueur cours je diminue l'endurance jusqu'à ce que l'endurance tombe à 0 ou que le joueur arrête de courir. Le joueur s'il n'a plus d'endurance arrête de courir et récupère petit à petit de l'endurance. Pour pouvoir utiliser l'actualisation du joueur dans les classes qui vont lui hériter je renomme `update` en `updatePlayer` et je l'appellerai dans chacune des classes qui hérite de `player`. Ensuite je rajoute un cube qui fera office de punchingball afin de tester les compétences offensives du personnage.

La capacité du guerrier est assez simple à implémenter. J'ai tout d'abord rajouté dans la classe `player` qui sera à l'origine de toutes les classes des joueurs les méthodes `abstract attack()`, `skill()` et `dealDamage()`, puis je crée une nouvelle classe qui hérite de `player` afin d'implémenter les méthodes spécifiques au guerrier.

9.1 Attack()

Je commence par l'attaque de base qui est un coup d'épée, je commence d'abord par récupérer à chaque `update` l'input du joueur et si il fait clique gauche alors je fais appel à la méthode `attack()`. Dans lequel je commence alors par créer un tableau des `Collider2D` dans un rayon d'une distance que je fixe à 7 pour l'instant mais qui changera sûrement. Un fois que j'ai ce tableau j'itère sur chaque élément en vérifiant si le tag de ces `rigidbody` est "ennemies" alors si c'est le cas je fais appel à la méthode `dealDamage()` de celui-ci avec une valeur fixe qui est stockée dans une variable privée `strength`. Par la suite je suis revenu sur cette méthode car elle posait plusieurs problèmes le premier était qu'à cause du fait que je cherche tous les `rigidbody` puis que je ne vérifie leurs tags que plus tard cela pose des problèmes d'optimisation car je récupère aussi les `rigidbody` des murs de la map par exemple. Je suis donc allé me renseigner dans la documentation de unity pour voir si il n'y avait un meilleur moyen et j'ai alors trouvé que l'on pouvait rajouter un paramètre sur la méthode `Physics2D.RaycastAll()` qui permet de ne vérifier que les `rigidbody` présents sur un certain layer (calque), on change alors de méthode en mettant les ennemis

sur un calque précis afin de ne récupérer que les ennemies de ce layer, je mets le layer en paramètre de play afin de pouvoir le récupérer plus facilement. Je peux alors retirer la vérification de tag. Ensuite il fallait que je rajoute un temps de chargement après avoir donné un coup d'épée. Pour ce faire j'ai créé un booléen qui agit comme un cadenas sur l'attaque qui ne s'effectue que s'il est à vrai. Ensuite je crée une méthode IEnumerator permettant de mettre ce booléen à faux puis d'attendre 2 secondes et de le remettre à vrai. J'appelle ensuite cette méthode à la fin de l'attaque et il y a maintenant un temps d'attente entre deux attaques du guerrier. Enfin le dernier problème et pas des moindres est le fait qu'en récupérant les rigidbody à l'aide d'un rayon, le guerrier fait des dégâts à tous les ennemies dans ce rayon ce qui comprend donc également les ennemies dans le dos de celui-ci. Je récupère donc la direction dans lequel regarde le joueur à l'aide du curseur de la souris, et je ne récupère alors seulement que les ennemies dans la direction du curseur.

9.2 Skill()

Pour le skill du guerrier je n'ai pas eu beaucoup de problème. Afin de créer une capacité capable de parer des coups je rends invincible le joueur durant une courte période. Je crée un booléen invincible que je mets à faux par défaut puis lorsque l'on appuie sur E la touche du sort alors le booléen devient vrai et dans la méthode dealdamage() du guerrier je crée une condition qui si le booléen invincible est vrai alors il ne prend pas de dégâts. Pour finir cette méthode je rajoute comme pour l'attaque une méthode IEnumerator qui crée un temps de rechargement plutôt long pour éviter de pouvoir en abuser. Je ne suis pas spécialement revenu sur cette méthode qui fonctionne très bien, pour l'équilibrage je reviendrais sûrement sur les temps d'invincibilité et de rechargement.

10 Multijoueurs

Pour le multijoueur j'ai très vite trouvé la méthode qui me permettrait de l'introduire sans devoir tout refaire de zéro. J'ai trouvé sur le marketplace de unity une bibliothèque qui s'appelle photon est permet de connecter ensemble jusqu'à 20 personnes dans le jeu ce qui est je pense largement suffisant, sachant que si l'on souhaite commercialiser le jeu il existe une option payante permettant d'augmenter le nombre de joueurs en payant.

10.1 Connection des joueurs sur un même serveur

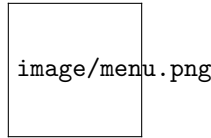
Pour commencer je crée tout le système permettant de connecter plusieurs joueurs sur le même serveur. Pour cela je crée un compte sur photon et crée un serveur. Une fois celui-ci créé je récupère l'app id qui est une clé permettant à unity de se connecter au serveur que je viens de créer. Une fois cela fait j'importe la bibliothèque dans le projet et indique mon app id. Puis je

créé dans un nouveau script NetworkManager une classe qui hérite de MonoBehaviour PunCallbacks une classe que j'importe depuis la bibliothèque photon. Cela permet de réduire considérablement la charge de travail pour le multi-joueur. Dans la méthode awake je me connecte à photon dans la région 'eu', cette connexion vas appeler la méthode OnConnectedToMaster(), dans cette méthode je rejoins le lobby du serveur puis dans la méthode suivant qui est appelée lorsque le lobby est rejoint alors on crée une salle avec un code. Pour ce faire je crée une méthode CreateRoom qui grâce aux méthodes intégrées dans photon crée ou rejoins la salle avec le nom "Room code" limité à 4 personnes puisque le jeu se jouera au maximum de 4. Cela permettra par la suite de créer une salle avec un code à donner à ces amis pour qu'ils puissent rejoindre

10.2 création d'un personnage par joueurs

Pour que chacun est son propre joueur on doit pouvoir instancier un nouveau joueur à chaque fois que quelqu'un rejoint la partie. Je rajoute donc le prefab du joueur dans les ressources de la bibliothèque photon puis je la mets en paramètres de NetworkManager. Je crée également un objet vide pour permettre aux joueurs de se connecter à un endroit précis de la map. Ce point de connexion doit aussi être ajouté dans la ressource de photon et ajouter dans les paramètres de NetworkManager. Une fois cela fait je fais en sorte qu'une fois la room créée j'instancie alors un nouveau joueur aux coordonnées donner en paramètres grâce à la méthode photonNetwork.Instantiate() je mets en paramètre le prefab du joueur ainsi que le transform de la position du point d'apparition des joueurs. Je teste et alors les personnages sont bien créés mais j'ai plusieurs problèmes tout d'abord c'est la caméra qui est unique aux deux joueurs et ne se déplace pas. Et le deuxième problème est que je ne filtre pas les entrées des joueurs, je déplace alors les deux joueurs en même temps. Un autre problème est la synchronisation de la position des joueurs qui ne se fait pas. Pour les problèmes de synchronisations c'est assez simple il existe en fait un component unity créé par photon qui actualise pour le serveur l'objet auquel il est lié, je rajoute donc ce component sur le prefab du joueur. Pour trier les entrées des joueurs je dois rajouter une condition dans le script du guerrier qui update et initialise le joueur seulement si c'est le sien pour le vérifier il suffit d'utiliser la méthode de photon gameObject.GetPhotonView().IsMine qui renvoie un booléen indiquant si le gameObject en paramètre est le sien ou pas. Enfin il me reste le problème de caméra pour commencer je supprime la caméra qui était sur la scène au début. Puis à l'initialisation dans la méthode "Start" de la classe Player je rajoute une variable contenant une nouvelle caméra que l'on initialise que l'on met sur orthographique et que l'on positionne au-dessus du joueur. Ensuite on déplace la caméra en même temps que le joueur mais à une distance fixe sur Z afin de toujours avoir une vue de dessus sur le joueur.

10.3 Menu de connexion



Maintenant que je peux connecter plusieurs joueurs sur la scène il est nécessaire de créer un menu permettant de choisir si l'on veut jouer en local ou en multijoueur et si l'on souhaite jouer en multijoueur indiquer le code de la salle que l'on souhaite créer ou rejoindre. Pour commencer je crée une nouvelle scène avec un bouton qui servira à la connexion en local, et une zone d'écriture où l'on pourra écrire le code de 4 caractères pour jouer en multijoueur. Ensuite je supprime le `NetworkManager` de la scène de jeu et le rajoute dans la scène du menu de connexion. Je rajoute dans cette classe une variable contenant le code de la salle une autre indiquant si le joueur est connecté ou pas et une autre indiquant si la scène a été load. Dans `update` si le joueur n'est pas connecté je vérifie si le joueur appuie sur entrée et alors je récupère le code rentré dans la zone de texte si il fait exactement 4 caractères et le stocke dans le code de la salle puis j'appelle la fonction `awake` que je renomme en `Awake1` pour pouvoir l'appeler quand je veux et non pas au début du script. Ce script permet d'initialiser le code de la salle à rejoindre ou créer.

Ensuite j'utilise les mêmes fonctions sans les modifier, je modifie simplement la méthode qui une fois la salle rejointe je load la scène du premier niveau fait par Jason. Pour load la scène j'utilise la méthode `SceneManager.LoadScene()` puis je modifie le booléen une fois que la scène a été load pour indiquer que la scène est load est que l'on peut la charger. Alors dans `update` je vérifie que le booléen soit à vrai puis je mets la scène load en scène active et je supprime tous les éléments de la scène pour rejoindre la partie et j'instancie le joueur. Grâce à ces modifications je suis maintenant capable de créer différentes salles et de choisir celles que je souhaite rejoindre.

Mais il reste un problème c'est que je ne peux pas jouer sans me connecter. Pour cela je crée un nouveau script qui s'activera seulement lorsque le bouton "jouer en local" sera appuyé. Dans ce script je reprends les mêmes méthodes permettant d'initialiser le joueur et de charger la map mais je retire toutes les méthodes qui relient le joueur au serveur.

11 L'archer

Finalement j'ai rajouté une nouvelle classe un petit peu plus complexe que le guerrier au jeu. Il s'agit de l'archer il a une attaque qu'il peut charger et qui selon le temps de chargement ira plus ou moins loin et fera plus ou moins de dégâts et il peut aussi utiliser une capacité qui permet de se déplacer beaucoup plus rapidement sur un très court laps de temps.

11.1 Attack()

Pour l'attaque basique de l'archer je dois à chaque update vérifier si le joueur appuis sur le clic de la souris, puis il fallait que je calcul une valeur celons le temps qu'il reste appuyer et pour cela je passe par les méthodes time de unity. Grace a cela je peux faire la différence entre le time quand j'appuie que je stock dans une variable puis je l'utilise et fait la différence avec le temps lorsque je lâche le clic de la souris. Mais cela implique plusieurs problème notamment le fait que la valeur pourrais être beaucoup trop importante si on reste appuyer et comme le temps agit comme un multiplicateur sur les dégâts de la flèche alors je décide de lorsque l'on relâche le clic de vérifier si il n'est pas au-dessus d'un certaine valeur et si il est au-dessus alors on change le multiplicateur de dégâts par le multiplicateur maximum que l'on a fixé.

Une fois que l'on sait comment calculer les dégâts que vas faire la flèche il faut créé l'objet puis l'initialiser. Je créé donc un nouveau prefab flèche que je rajoute aussi au ressource de Photon pour que chaque joueur vois bien les même flèches. Puis a cet objet de flèche je rajoute un script qui a chaque update avance vers une direction que je mets un paramètre. La vitesse est la même peut importe la puissance donc je créé un déplacement grâce a une valeur arbitraire. Puis je créé une valeur d'existence qui est instancier au début celons la puissance de la flèche et qui décrémenter au fur et à mesure du temps.

Ensuite je retourne sur l'archer et je rajoute l'initialisation d'une nouvelle flèche lorsque l'on lâche le clique celons et l'initialisation de la direction et de la puissance de celle-ci. Je rajoute qu'a chaque actualisation la flèche vérifie si elle touche quelque chose. Si elle touche quelque chose alors elle est supprimée et si elle touche une ennemie alors il lui fait des dégâts celons sa puissance. Pour rajouter un peu de d'effet de chargement je diminue la vitesse du joueur lorsqu'il charge la flèche et lui redonne la vitesse lorsqu'il a fini.

11.2 Skill()

Pour la compétence de l'archer ce n'était pas très compliquer je lui donne un boost de vitesse durant une courte durée grâce à un IEnumerator et empêche le fait qu'il puisse courir pendant la compétence. Je lui enlève aussi une bonne partie de son endurance et un temps de chargement pour éviter que l'on puisse en abusée. L'archer est donc maintenant pleinement implémenté.

12 Mes impressions

J'ai trouver que le début de ce projet était une bonne expérience et permet de bien mettre en œuvre nos connaissance que l'on accumule au fur et a mesure en programmation. Mais j'ai malgré tout eu l'impression d'être abandonner par plusieurs de mes camarades ce qui a rendu le début du projet un petit peu

compliquer, maintenant que cela est passer je pense que nous sommes bien lancés et que nous allons dans la bonne direction.