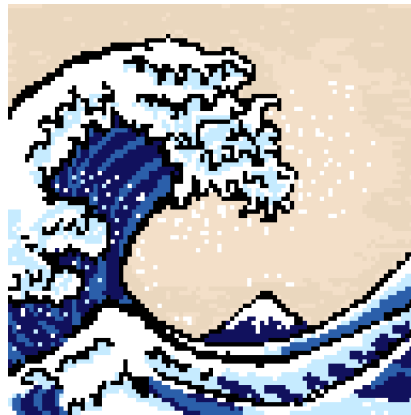


Rapport de projet Hokusai no Game

Groupe OUI : Jason, Rémi et Arthur

7 Juin 2023



Contents

I	Introduction	4
1	Notre groupe	4
2	Origine et nature du projet	4
3	Objet de l'étude	4
3.1	Nos objectifs	4
4	État de l'art	5
4.1	Histoire du roguelike	5
4.2	Inspirations et points forts	5
5	A propos du jeu	5
5.1	Contexte du jeu	5
5.2	Caractéristiques du jeu	6
6	Répartition	6
7	Avancement	7
II	Jason	7
8	Caractéristiques des ennemis	7
8.1	Kaibutsu Walker	7
8.2	Enemy Animations	10
8.3	Kaibutsu Lezard	12
8.4	Système de vague	13
8.4.1	Zones de spawn	14
8.4.2	Script du spawner	14
8.5	Le boss	15
9	Les maps	18
10	Ressenti	22
III	Rémi	22
11	Chronologie de Rémi	22
11.1	Le Mage (les debuts)	22
11.2	Le site (les debuts)	23
11.3	Système d'amélioration (les debuts)	25
11.4	Le Mage (la suite)	26

11.5 Le site (la suite)	27
11.6 L'assassin	29
11.7 Le menu de fin de jeu (perdu)	30
11.8 Correction de dysfonctionnement	31
11.9 Le Mage (fin)	32
11.10Le site internet(fin)	32
11.11 Systeme d'amélioration (fin)	33
11.12 Mes impresions	35
 IV Arhur	 36
12 Caractéristiques des personnages	36
12.1 Classes des personnages	36
12.2 Le guerrier	36
12.3 L'archer	37
 13 Multijoueurs	 39
13.1 Connection des joueurs sur un même serveur	39
13.2 création d'un personnage par joueurs	39
13.3 Menu de connection	40
13.4 Mise en commun des pv des monstre	41
13.5 Mise en commun de la map et du changment de map selon le propriétaire du serveur	42
 14 Animation Archer et Guerrier	 43
14.1 Selection des assets	43
14.2 Importation dans Unity	43
14.3 Animation de l'archer	44
 15 Interface	 44
15.1 Lancement d'une partie	44
15.2 Menu de pause	45
15.3 Menu de selection d'amélioration des statistiques	45
15.4 Menu de la carte et passage au prochain niveau	46
15.5 Menu de tutoriel	48
 16 Sound effect	 49
 17 L'installateur et le désinstallateur	 49
 18 Mes impressions	 50

Part I

Introduction

1 Notre groupe

Le projet OUI est composé de 4 étudiants de première année de la classe D2 : Arthur LEFEBVRE, Jason QIU et Rémi OMS. Notre équipe s’est formée grâce à une passion commune : celle de l’animation japonaise et tout ce qui englobe cette dernière.

2 Origine et nature du projet

En commençant à réfléchir à notre projet, nous avons tout de suite penser à nous orienter vers un jeu avec une direction artistique en pixel art, fortement inspirée des jeux rpg types Pokemon/ Octopath Traveler. En parallèle, nous étions tous très attirés par les jeux proposant un réel challenge, pour ne pas citer *Elden Ring*, *Isaac*, *Enter the Gungeon* ou encore la série des Dark Souls. Mais pour ne pas tomber dans le cliché du jeu de survie générique, nous étions en quête d’un argument, d’une particularité qui rendrait notre jeu plus ”unique”. C’est pourquoi nous étions déterminés à inclure une composante coopérative à notre jeu. Depuis le début, nous voulions que nos joueurs évoluent ensemble en unissant leurs forces et leurs atouts. Petit à petit, le projet a donc prit la forme d’un Rogue Like coopératif, englobé par un scénario entouré de plein de références à l’univers d’internet et de la pop culture.

En espérant que notre jeu plaira, nous sommes fiers de vous présenter notre projet, Hokusai No Game.

3 Objet de l’étude

3.1 Nos objectifs

L’objectif principal est de réussir à créer un jeu avec une bonne rejouabilité, capable de divertir et de faire passer un bon moment à nos joueurs. Mais nous avons aussi pour objectifs de progresser dans l’utilisation des divers outils que l’on sera amenés à utiliser. Que ce soit Unity, Blender, Audacity, Photoshop ou tout autre outil créatif. Ce projet est l’occasion pour nous d’acquérir des compétences dans des domaines que nous ne maîtrisions pas forcément comme le dessin ou la musique et d’augmenter nos compétences en programmation.

Plus largement encore, ce projet est surtout l’occasion pour nous d’approfondir nos connaissances en programmation en mettant en pratique tout ce que l’on apprend au cours de notre année de SUP. Il est essentiel pour nous d’améliorer nos capacités de compréhension et de réflexion en général. Un des aspects primordiaux de ce projet est aussi d’acquérir une bonne organisation. Le but est de

répartir équitablement les tâches entre nous 3 dans le but d'être le plus efficace possible.

En bref, notre objectif final reste avant tout de prendre du plaisir. Nous souhaitons avant tout produire un jeu qui plaît et qui nous satisfait. Ce projet est l'occasion pour nous d'acquérir les fondamentaux du travail de l'ingénieur en entreprise. Cet avant-goût du travail de l'ingénieur en informatique ne peut être que bénéfique pour nos futurs projets.

4 État de l'art

4.1 Histoire du roguelike

Le roguelike prend ses origines avec comme son nom l'indique le jeu *Rogue* en 1980, l'objectif de ce jeu est d'explorer des sous terrains pour y trouver *l'Amulette de Yendor*. L'argument principal de *Rogue* est sa rejouabilité, en effet le joueur ne dispose dans ce jeu que d'une seule vie ce qui rend l'exploration plus compliquée. Si le personnage meurt alors le joueur doit recommencer depuis le début. Mais la subtilité vient surtout dans la génération procédurale de la carte qui permet de donner une bonne rejouabilité au jeu sans le rendre trop chronophage.

Par la suite de nombreux jeux se sont inspirés de *Rogue* pour former le genre roguelike, les premiers à s'en inspirer sont *NetHack* (1987), *Angband* (1990), *Ancient Domains of Mystery* (1994) et *Donjon Mystère* (1993).

4.2 Inspirations et points forts

Nos principales inspirations sont les jeux comme *Hotline Miami*, *Enter The Gungeon* ou encore *Endless waves survival* ce dernier est moins connu mais nous sommes tombés dessus en cherchant des inspirations et les graphismes simples mais efficaces nous ont inspirés à donner cette direction artistique à notre jeu.

Les principaux points forts de ces jeux sont le dynamisme de leurs systèmes de combat et l'efficacité de leur direction artistique de type pixel art qui est agréable sans être trop complexe.

Les avantages et principales caractéristiques de notre jeu doivent donc être:

- Un gameplay satisfaisant et dynamique.
- Une difficulté bien dosée.
- Une customisation de la progression grâce aux choix des joueurs

5 A propos du jeu

5.1 Contexte du jeu

La Terre est envahie par des monstres intelligents (les Kaibutsus) qui après avoir analysé le comportement des humains à l'aide d'Internet ont décidé de prendre

l'apparence et les caractéristiques des créatures les plus terrifiantes qu'ils ont trouvés. Un petit groupe de Nightmare-city est déterminé à sauver le monde et pour ce faire, ils devront traverser la ville afin de trouver et terrasser le boss ultime: "le Kaiju".

5.2 Caractéristiques du jeu

L'objectif de Hokusai no game est donc de traverser la ville (la carte) et d'atteindre la dernière case du jeu pour battre "le Kaiju". Cependant la difficulté du jeu réside dans le fait que le joueur ne possède qu'une seule vie, et que s'il meurt au cours de la partie il devra recommencer depuis le début. Cela en fait un parfait jeu pour le speed-run étant donné que le jeu sera plus ou moins court selon les choix fait par les joueurs se qui permettra aux joueurs plus expérimentés de faire des choix plus risquer afin de finir le jeu le plus rapidement possible.

6 Répartition

Tache	Rémi	Arthur	Jason
Mouvements et compétences de base des personnages		O	
Compétences du guerrier		O	
Compétences de l'archer		O	
Compétences du mage	O		
Compétences de l'assassin	O		
Mouvements et attaques des kaibutsu-walker			O
Mouvements et attaques des kaibutsu-lezard			O
Mouvements et attaques du kaibutsu primordial			O
Animations	O	O	O
Sons et musiques		O	
Système d'amélioration des statistiques		O	
Système d'amélioration des compétences	O		
Système de vague et sélection des récompenses en cas de victoire		O	
Nourriture lâchée par des monstres	O		
Déplacements entre les différentes cases de la carte		O	
Aperçu de la carte		O	
Création de toutes les cases différentes			O
Multijoueur coopératif		O	
Choix des classes au début de la partie		O	
Menu pour lancer une partie		O	
Écrans de défaite et de victoire	O		
Menu de pause et affichage des statistiques/compétences		O	
Site internet	O		

7 Avancement

Le tableau ci-dessous répertorie toutes les tâches et leurs avancements en pourcentage selon les soutenances :

Tache	Mars	Avril	Juin
Mouvements et compétences de base des personnages	100	//	//
Compétences du guerrier	100	//	//
Compétences de l'archer	50	100	//
Compétences du mage	25	100	//
Compétences de l'assassin	0	50	100
Mouvements et attaques des kaibutsu-walker	100	//	//
Mouvements et attaques des kaibutsu-lezard	0	100	//
Mouvements et attaques du kaibutsu primordial	0	0	100
Animations	0	50	100
Sons et musiques	0	100	//
Système d'amélioration des statistiques	0	100	//
Système d'amélioration des compétences	0	0	100
Système de vague et sélection des récompenses en cas de victoire	0	100	//
Nourriture lâchée par des monstres	100	//	//
Déplacements entre les différentes cases de la carte	0	50	100
Aperçu de la carte	0	50	100
Création de toutes les cases différentes	10	50	100
Multijoueur coopératif	50	50	100
Choix des classes au début de la partie	0	100	//
Menu pour lancer une partie	100	//	//
Écrans de défaite et de victoire	0	50	100
Menu de pause et affichage des statistiques/compétences	25	50	100
Site internet	50	75	100

Part II

Jason

8 Caractéristiques des ennemis

8.1 Kaibutsu Walker

Les ennemis qui sont en plus grand nombre sont les Kaibutsu walker, c'est-à-dire des Kaibutsu qui ont pris l'apparence de zombies et ne font que marcher et mordre. Ils se déplacent plutôt lentement et font peu de dégâts mais ils

apparaissent en grand nombre et peuvent être redoutables lorsqu'ils nous foncent tous dessus. Ils ont un nombre de points de vie élevé.

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.

Lors de la poursuite du joueur par le zombie, à chaque frame, un nouveau waypoint est créé pour représenter la position actuelle du joueur. Cela est nécessaire car les waypoints sont normalement créés uniquement après que le zombie s'y soit rendu. Cependant, lorsqu'un ennemi commence à pourchasser un joueur, le waypoint précédent n'est pas actualisé, ce qui peut entraîner un déplacement inefficace lorsque le zombie cesse sa poursuite. En effet, il peut être amené à effectuer une longue ligne droite pour retourner au dernier waypoint créé avant la poursuite. Afin d'éviter ce comportement non réaliste et d'améliorer le déplacement du zombie, j'ai modifié la logique pour actualiser le waypoint à chaque frame pendant la poursuite. Ainsi, le waypoint représente toujours la position actuelle du joueur, garantissant que le zombie suit le joueur de manière plus précise et réaliste. Cette approche permet au zombie de s'adapter en temps réel aux mouvements du joueur et de prendre des décisions plus intelligentes pour le poursuivre. Le résultat est un comportement de poursuite plus fluide et plus naturel pour le zombie.

Par la suite, j'ai amélioré le système de poursuite du zombie pour le rendre compatible avec le multijoueur. Dans la première version de la méthode ChasePlayer(), qui permet à un ennemi de poursuivre un joueur à proximité, j'utilisais le GameObject Player pour obtenir la distance du joueur à tout moment, ce qui me permettait de décider quand le zombie devait commencer à le poursuivre. Cependant, cette approche ne fonctionnait pas correctement lorsque plusieurs joueurs étaient à proximité, car cela créait des problèmes pour le zombie qui ne savait pas quel joueur poursuivre parmi les deux. Pour résoudre ce problème, j'ai ajusté la logique de poursuite pour prendre en compte plusieurs joueurs. Au lieu d'utiliser directement le GameObject Player, j'ai utilisé une liste de GameObjects pour représenter tous les joueurs à proximité. À chaque frame, j'ai parcouru cette liste pour trouver le joueur le plus proche du zombie et j'ai poursuivi ce joueur spécifique.

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.

Pour garantir un réalisme dans le jeu, j'ai rencontré des problèmes liés aux déplacements des ennemis à travers les obstacles et les bâtiments. Pour résoudre cela, j'ai ajouté un `Collider2D` et un `Rigidbody2D` à l'ennemi. J'ai configuré le `Rigidbody2D` pour qu'il n'ait pas de gravité et verrouillé l'axe `Z` pour empêcher le zombie de basculer.

Cependant, cela a entraîné un autre problème lorsqu'un joueur essayait de pousser le zombie. L'ennemi s'envolait dans la direction de la poussée, ce qui brisait complètement le réalisme du jeu. Pour remédier à cela, j'ai dû trouver une solution. J'ai essayé de verrouiller les axes `X` et `Y` pour empêcher le zombie d'être poussé, mais cela n'était pas ce que je souhaitais. De plus, en faisant cela, le zombie pouvait traverser les murs sans raison apparente. À l'origine, j'utilisais le composant `Transform` pour déplacer l'ennemi. Cependant,

j'ai réalisé que je devais utiliser le Rigidbody2D plutôt que le Transform pour résoudre ce problème. C'est alors que j'ai découvert la méthode MovePosition de la classe Rigidbody2D, qui permet de déplacer un Rigidbody2D en fonction d'un vecteur donné. En utilisant la méthode MovePosition, j'ai pu mettre à jour la position du Rigidbody2D de l'ennemi en fonction des interactions avec le joueur. Cela a permis au joueur de pousser le zombie sans qu'il s'envole à l'infini. De plus, en utilisant la physique du Rigidbody2D, le zombie n'était plus en mesure de traverser les murs de manière non réaliste. Cette approche a permis de concilier les mouvements du joueur et du zombie de manière réaliste, tout en maintenant l'intégrité des collisions avec les obstacles et les bâtiments.

8.2 Enemy Animations

Pour l'animation de l'ennemi, j'ai utilisé une fonctionnalité de Unity appelée Blend Tree, qui m'a permis d'implémenter les animations de marche du Kaibutsu Walker (l'ennemi de base) sans avoir à écrire une grande quantité de code.

L'ennemi dispose de quatre animations de marche : vers le haut, vers le bas, vers la gauche et vers la droite. Il est assez compliqué de mettre en place la bonne animation en fonction du déplacement de l'ennemi uniquement avec du code. Cependant, grâce à l'outil Blend Tree de Unity, il est possible d'assigner facilement une animation en fonction du déplacement du zombie.

Voici plus en détail comment j'ai procédé :

Tout d'abord, pour synchroniser les animations avec le déplacement de l'ennemi, il est nécessaire de connaître la direction dans laquelle il se déplace à chaque frame. Pour cela, j'ai calculé la direction en soustrayant sa position actuelle de sa position précédente, qui est également calculée à chaque frame. Cela nous donne un Vector2 représentant la direction.

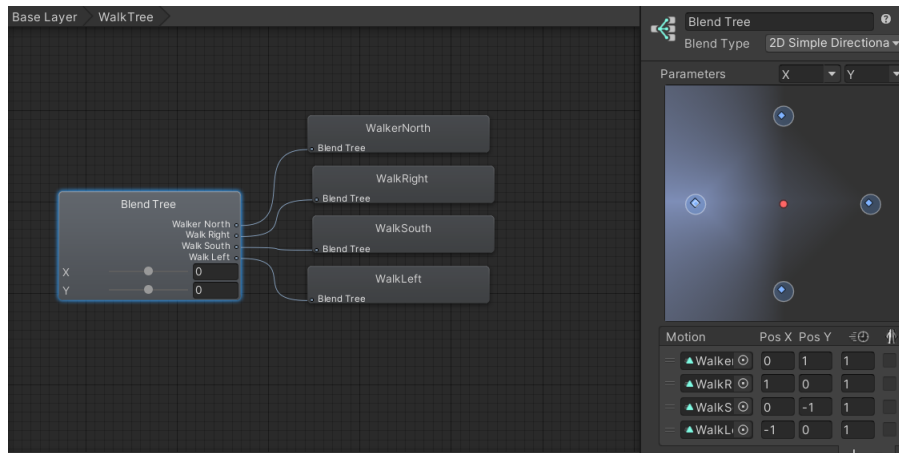
Ensuite, j'ai créé deux valeurs float, X et Y, qui sont les composantes utilisées pour changer les animations de marche lorsque cela est nécessaire. Ces valeurs se situent entre -1 et 1.

La composante X représente la direction horizontale et la composante Y représente la direction verticale. Ensuite, il suffit de configurer les bonnes valeurs de X et Y dans le Blend Tree et d'associer les animations correspondantes à chaque direction.

Unity se charge alors automatiquement d'afficher la bonne animation en fonction des valeurs de X et Y.

En utilisant le Blend Tree, j'ai pu simplifier considérablement la gestion des animations de marche de l'ennemi. Cela m'a permis d'éviter d'écrire de multiples conditions dans le code pour changer les animations en fonction de la direction du déplacement. L'outil Blend Tree de Unity s'occupe de manière automatique de l'affichage de la bonne animation en fonction des valeurs spécifiées.

Cette approche facilite grandement la gestion des animations, en permettant d'attribuer les bonnes animations aux différentes directions de déplacement de l'ennemi, sans avoir à écrire un code complexe et répétitif.



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.

Universal LPC Spritesheet Generator

Free/Libre pixel art sprites from the [Liberated Pixel Cup](https://liberatedpixelcup.com/) and [OpenGameArt.org](https://opengameart.org/).
License: [CC-BY-SA 3.0](https://creativecommons.org/licenses/by-sa/3.0/). You must [credit the authors of this artwork](#).

Download: [Image \(PNG\)](#) [Credits \(text\)](#) [Credits \(CSV\)](#)

Import/Export: [Export to Clipboard \(JSON\)](#)

Edit: [Reset all](#) [Replace Mask \(pink\)](#)

View: [Collapse All](#) [Expand Selected](#) ☐ Compact Display

Search:

Body
 Body type ▶
 Shadow ▶
 Body color ▶
 Special ▶
 Zombie ▶
 Skeleton ▶
 Wounds ▶
 Arm ▶
 Brain ▶
 Ribs ▶
 Eye ▶
☐ No wound_eye ☒ eye
 Mouth ▶
 Prostheses ▶
 Lizard ▶
 Head
 Heads ▶
 Human female ▶
 Human male ▶
 Human female elderly ▶
 Human male elderly ▶
 Boorman ▶
 Pig ▶
 Sheep ▶
 Minotaur ▶
 Minotaur female ▶
 Wartotaur ▶
 Wolf female ▶
 Wolf male ▶
 Rabbit ▶
 Rat ▶
 Mouse ▶
 Lizard female ▶
 Lizard male ▶
 Orc female ▶
 Orc male ▶
 Goblin ▶
 Alien ▶

Preview [Walk](#)



Complete sprite sheet (click to zoom):



Pour créer les animations, j'ai importé tous les spritesheets des animations, ajouté chaque animations 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.

8.3 Kaibutsu Léopard

Le dernier type de Kaibutsu que l'on retrouve sont les chiens cracheurs de venin, ils attaquent les joueurs en crachant du venin sur les joueurs, ils tirent de loin mais ont peu de points de vie.

Un autre gros ajout que j'ai fait est un deuxième type d'ennemi: le Kaibutsu Léopard. A la base cela devait être un chien mais comme je n'ai pas trouvé d'assets de chiens en 2D qui rendait bien avec le jeu, j'ai opté pour un ennemi fait avec le site Universal LPC sprite sheet character generator qui a l'apparence d'un léopard humanoïde qui attaque à distance en lançant des boules de feu.



J'ai créé 3 autres scripts Csharp : LezardAnimations, LezardBehaviour et LezardMovements qui dérivent respectivement de leurs parents : EnemyAnimations, EnemyBehaviour et EnemyMovements.

Pour le script LezardAnimations, j'ai mis exactement la même chose que le zombie, ses animations de marche et de mort sont similaires.

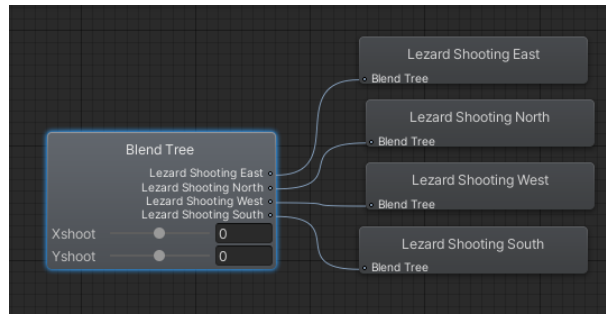
Pour le script LezardBehaviour, il n'a pas la même attaque que le zombie, son attaque sera défini dans LezardMovements pour plus de simplicité. La méthode Start() reste pareil, mais dans Update, il y a seulement la méthode Death() qui est actualisée pour savoir si le léopard est mort ou non.

C'est dans le script LezardMovements qu'il y a les modifications majeures concernant le comportement du nouvel ennemi. J'ai modifié la méthode ChasePlayer() dans EnemyMovements et je l'ai passé en méthode virtuelle. Cela veut dire que la méthode ChasePlayer() dans EnemyMovements est la méthode par défaut mais que je peux changer la méthode en redéfinissant la méthode pour un autre type d'ennemi comme le léopard. Ici j'ai fait en sorte que le léopard ne se rapproche d'un joueur seulement s'il est à plus d'une certaine distance de lui

et si la distance entre le lézard et le joueur le plus proche est inférieure à la distance seuil, il commencera alors à reculer pour prendre de la distance.

Ensuite j'ai aussi modifié la méthode `Update()`; si le lézard voit un joueur alors il le poursuit ou recule en fonction de la distance qui les sépare puis s'il n'est pas déjà en train de lancer une boule de feu, cela va trigger l'animation de l'invocation de la boule de feu grâce a la méthode `SetTrigger`. L'animation d'invocation de la boule de feu a été importée grâce la feuille d'animation déjà générée par le site.

Pour l'animation d'invocation de la boule de feu, j'ai utilisé la même méthode que pour ses animations de marche : un blend tree qui gère quelle animation jouer en fonction de la direction dans laquelle il va lancer sa boule de feu.



A la fin de l'animation de l'invocation de la boule de feu, je lance la méthode `SpawnFireball()` en ajoutant une animation event a la dernière frame d'animation d'invocation.

Dans la méthode `SpawnFireball()`, j'appelle la méthode `Shoot()` qui fait apparaître la boule de feu. Dans la méthode `Shoot()`, je gère les coordonnées `Xshoot` et `Yshoot` qui permettent de régler l'animation d'invocation correctement. Je calcule aussi l'angle de rotation pour que la boule de feu apparaît orientée dans la direction du joueur le plus proche grâce a la méthode `Atan2` de `Mathf`. Je fais apparaître la boule de feu un au dessus du lézard pour synchroniser avec l'animation d'invocation.

8.4 Systeme de vague

Notre jeu se base sur un système de vagues, c'est a dire que a chaque étage un certain nombre d'ennemies est générer dans chaque case a des endroit prédéfinies.

Les joueurs devront alors choisir entre foncer dans le tas en alertant tout les Kaibutsu pour tous les abattre plus rapidement. Ou alors les joueurs pourrons choisir d'éviter de se faire voir par les Kaibutsu et de les attaquer par surprise. Dans tout les cas le but des joueurs est de battre tout les ennemis sur la carte afin d'avoir accès à une récompense et de pouvoir continuer le jeu.

8.4.1 Zones de spawn

Pour la deuxième soutenance, j'ai développé un système de spawn d'ennemis. J'ai créé un nouveau script C que j'ai appelé "Spawner". L'objectif était de faire apparaître des zombies à des emplacements aléatoires de la carte, tout en veillant à ce qu'ils n'apparaissent pas à l'intérieur des bâtiments ou des obstacles.

J'ai mis en place un système de zones d'apparition, que j'ai appelé "Spawn zones". Pour cela, j'ai créé des objets 2D rectangulaires dans Unity et je les ai placés à différents endroits de la carte. Ensuite, j'ai désactivé le composant "Sprite Renderer" de ces objets pour qu'ils ne soient pas visibles en jeu.

J'ai regroupé tous ces objets de zones d'apparition sous un objet parent, que j'ai appelé "SpawnZones". Cela permet de mieux organiser les zones et de les manipuler plus facilement dans le script.

Grâce à ce système de zones d'apparition, je peux maintenant faire apparaître des zombies aléatoirement sur la carte en sélectionnant une zone parmi celles disponibles. Cela garantit que les zombies n'apparaissent pas à des endroits inappropriés, tels que l'intérieur des bâtiments ou à travers des obstacles.

Cette approche offre une flexibilité dans la génération des ennemis et permet de créer une expérience de jeu plus intéressante, avec des zombies apparaissant de manière variée et imprévisible sur la carte.

8.4.2 Script du spawner

Dans le script C, j'ai créé une méthode appelée "SpawnEnemies()" en utilisant un IEnumerator. Dans cette méthode, je génère une position de spawn aléatoire en utilisant une autre méthode que j'ai créée : "GetRandomSpawnPosition()". Cette méthode permet de générer une position aléatoire sur la carte en choisissant une zone de spawn parmi toutes les zones d'apparition disponibles.

Pour obtenir une zone de spawn de manière aléatoire, j'accède d'abord à l'objet parent "SpawnZones". Ensuite, j'utilise la méthode "GetChild()" pour accéder à un enfant spécifique de cet objet. J'utilise également la fonction "Random.Range()" pour obtenir un index aléatoire parmi les zones de spawn disponibles.

Une fois que la zone de spawn a été choisie, j'utilise à nouveau des fonctions aléatoires pour générer une position aléatoire à l'intérieur de cette zone. Cela garantit que les ennemis apparaissent à des emplacements variés à l'intérieur de la zone sélectionnée.

Enfin, j'utilise une boucle "for" pour générer un certain nombre d'ennemis souhaité. Ce nombre peut varier en fonction des niveaux du jeu. J'appelle la méthode "SpawnEnemies()" dans la fonction "Start()" en utilisant la fonction "StartCoroutine()". Cela permet de lancer le spawn des ennemis dès le début du niveau et comme il s'agit d'une coroutine, je peux interrompre le spawn à tout moment pendant la partie.

En utilisant cette approche, je peux générer des ennemis de manière aléatoire à différents emplacements de la carte, tout en contrôlant le nombre d'ennemis et en garantissant qu'ils n'apparaissent pas dans des zones inappropriées.

8.5 Le boss

Pour la dernière soutenance, j’ai travaillé sur l’implémentation du boss, également appelé "Nightboss" ou "Boss de la nuit" en français. Ce boss est représenté par un chevalier noir et constitue un défi redoutable pour les joueurs. Il possède un nombre élevé de points de vie et inflige des dégâts considérables avec son épée. Cependant, il n’a pas d’attaque à distance, ce qui le rend vulnérable aux attaques des mages et des archers.

Le boss est conçu pour être un adversaire redoutable pour les guerriers et les assassins, car il peut engager le combat au corps à corps et infliger des dégâts significatifs. Cela incite les joueurs à adopter différentes stratégies en fonction de leur classe et de leurs capacités.

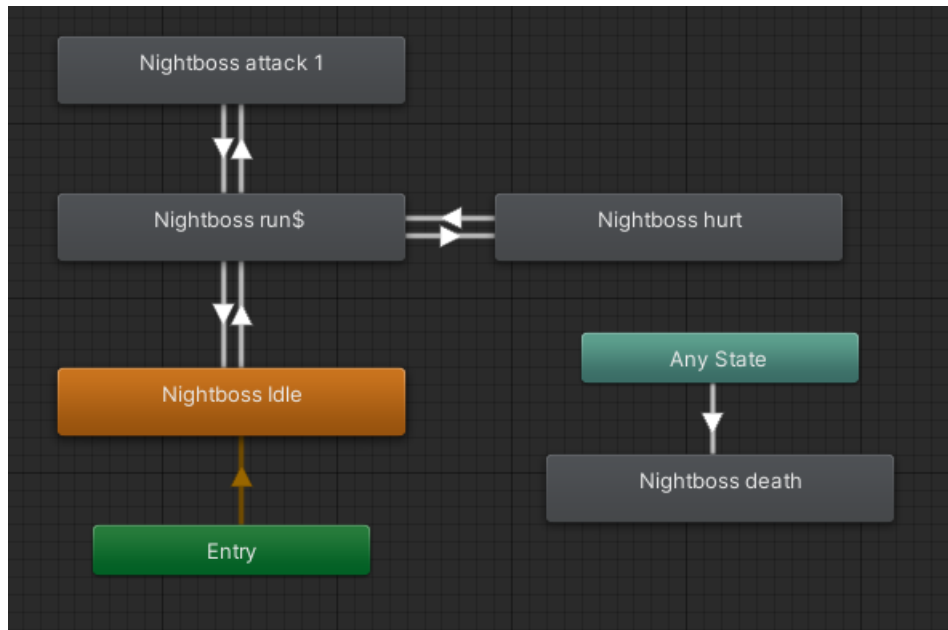
La conception du boss comprend également des mécanismes pour gérer ses points de vie, ses attaques et ses patterns de mouvement. Il peut avoir des phases différentes au cours du combat, où il peut devenir plus agressif ou utiliser des compétences spéciales pour augmenter le défi pour les joueurs.

L’objectif principal de l’implémentation du boss est de fournir une expérience de combat engageante et stimulante pour les joueurs. Sa puissance et ses capacités uniques en font un adversaire mémorable et un défi à surmonter pour les joueurs les plus expérimentés.

En résumé, le boss, également connu sous le nom de "Nightboss" ou "Boss de la nuit", est un adversaire imposant avec une apparence de chevalier noir. Il possède une grande quantité de points de vie et inflige des dégâts considérables avec son épée. Bien qu’il n’ait pas d’attaque à distance, il représente un défi pour les guerriers et les assassins, tandis que les mages et les archers peuvent profiter de leur avantage à distance. L’implémentation du boss comprend des mécanismes pour gérer sa santé, ses attaques et ses patterns de mouvement, offrant ainsi une expérience de combat captivante pour les joueurs.



Pour implémenter le boss dans Unity, j’ai utilisé principalement l’Animator Controller, également appelé State Machine Controller. Cet outil s’est avéré très efficace pour créer un boss avec des patterns de comportement précis.



J'ai commencé par créer une animation d'Idle, qui est l'animation de base lorsque le boss est immobile. Ensuite, j'ai configuré un mécanisme de détection de la présence d'un joueur dans le rayon du boss. Lorsqu'un joueur est détecté, un booléen appelé "IsRunning" est activé, ce qui déclenche la transition vers l'état "NightbossRun". Le boss se met alors à courir vers le joueur, passant de l'état Idle à l'état de course.

Si le boss se rapproche suffisamment d'un joueur, il déclenche une attaque en activant un paramètre de type "trigger", ce qui lance l'animation d'attaque. À la fin de l'animation d'attaque, le boss retourne à son état de course-poursuite. Cela permet de créer une séquence fluide où le boss poursuit les joueurs et les attaque lorsqu'ils sont à portée.

Le boss possède également une animation de mort qui se déclenche lorsque ses points de vie atteignent 0 ou moins. À la fin de cette animation, j'ai programmé la suppression du GameObject du boss pour le retirer de la scène.

L'utilisation de l'Animator Controller et de la State Machine permet de gérer facilement les différentes animations et états du boss, en créant des transitions fluides entre eux. Cela donne au boss une apparence réaliste et des comportements adaptés à chaque situation, rendant les combats contre lui plus immersifs et passionnants pour les joueurs.

J'ai créé un script Csharp qui permet de gérer l'orientation du boss en fonction du joueur qu'il poursuit. Je crée une méthode LookAtPlayer qui prend en paramètre un gameobject, ici celui du joueur et ensuite le boss va s'orienter en fonction de ce joueur.

Dans l'état "NightbossRun" de l'Animator Controller, j'ai créé un script pour gérer le déplacement du boss. Dans la méthode "OnStateEnter", qui est appelée une fois lorsque l'état "NightbossRun" est activé, j'ai récupéré le joueur en utilisant son tag et le Rigidbody du boss. J'ai également obtenu une instance du script "Nightboss" qui contient ma fonction "LookAtPlayer".

Dans la méthode "OnStateUpdate", qui est appelée à chaque frame entre "OnStateEnter" et "OnStateExit", j'ai mis à jour l'orientation du boss en appelant la fonction "LookAtPlayer" avec le joueur récupéré précédemment. Ensuite, j'ai créé un nouveau Vector2 qui représente la direction dans laquelle le boss se déplacera.

J'ai utilisé la méthode "MovePosition" du Rigidbody2D pour déplacer le boss selon la direction calculée. Ensuite, j'ai vérifié si la distance entre le joueur et le boss était inférieure à la distance d'attaque. Si c'était le cas, j'ai activé l'animation d'attaque en utilisant la méthode "SetTrigger".

Cette approche permet de contrôler le déplacement du boss vers le joueur dans l'état "NightbossRun". La fonction "LookAtPlayer" assure que le boss fait toujours face au joueur, tandis que la vérification de la distance d'attaque permet de déclencher l'animation d'attaque lorsque le joueur est à portée. Cela ajoute du réalisme et de l'interactivité aux combats contre le boss.

Pour la zone d'attaque du boss, j'ai créé deux carrés qui sont des objets prédéfinis de Unity et je les ai positionnés à gauche et à droite du boss. Ensuite, pour déterminer quelle zone d'attaque activer en fonction de la direction du boss, j'ai récupéré la variable "IsFlipped" et vérifié si elle est vraie ou fausse. En fonction de cela, j'active la hitbox correspondante en créant un Collider2D appelé "hitbox" et en lui attribuant la zone de gauche ou de droite. Une fois cela fait, j'ai utilisé la méthode "OverlapBoxAll" pour récupérer tous les Collider2D présents dans la zone d'attaque qui ont le masque "Action". Ensuite, j'ai vérifié si ces Collider2D sont des joueurs en comparant leur tag à l'aide de la méthode "CompareWithTag". Enfin, j'ai appliqué des dégâts aux joueurs en utilisant la méthode "DealDamage" en fonction de leur classe respective. Cela permet au boss d'infliger des dégâts aux joueurs se trouvant dans sa zone d'attaque.

Pour empêcher le boss de se déplacer pendant son attaque, j'ai créé un autre script lié à la machine d'états de l'attaque. Dans la méthode OnStateEnter, j'ai mis le Rigidbody en mode cinématique en définissant la propriété isKinematic sur True. Cela permet de désactiver les effets de la physique sur le boss, l'empêchant ainsi de bouger pendant son attaque. Ensuite, dans la méthode OnStateExit, j'ai rétabli le mode non cinématique du Rigidbody en définissant la propriété isKinematic sur False. Cela permet de permettre au boss de se déplacer à nouveau une fois son attaque terminée. Ces manipulations du mode cinématique du Rigidbody permettent de contrôler le comportement du boss pendant son attaque et de le figer temporairement pour qu'il ne puisse pas bouger, assurant ainsi que son mouvement est restreint à l'animation d'attaque.

9 Les maps

En tant que responsable de la création des maps dans notre jeu pixel art 2D, j'ai décidé d'utiliser le système de tilemap pour sa simplicité et sa rapidité de mise en place. Cependant, j'ai rencontré quelques difficultés au départ pour comprendre son fonctionnement, ce qui m'a pris beaucoup de temps.

Pour notre jeu qui se déroule dans une ville post-apocalyptique, il était crucial de créer une ambiance pesante en utilisant des assets appropriés. J'ai donc consacré de nombreuses heures à la recherche d'un asset de ville qui correspondrait parfaitement à l'atmosphère du jeu. Cependant, il était important de choisir un asset qui n'était pas trop détaillé en termes de relief, afin de simplifier la création des maps.

Finalement, j'ai trouvé un asset gratuit sur le site OpenGameArt qui répondait à nos besoins. J'ai téléchargé cet asset, qui était composé d'images au format PNG. Ensuite, j'ai entrepris de les configurer en tant que Tiles utilisables dans Unity.

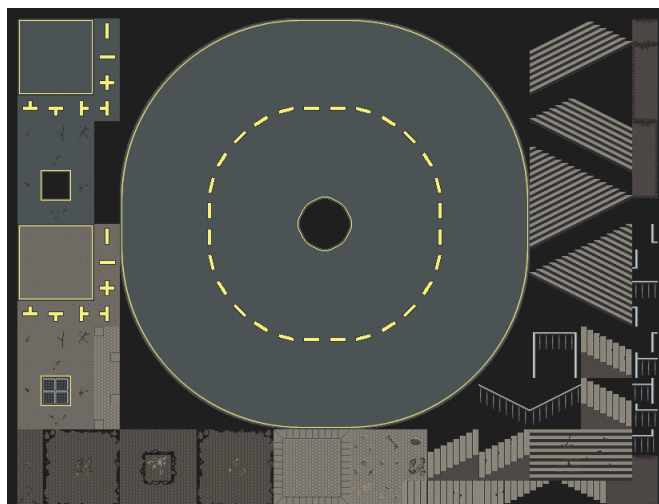
La première étape a été de modifier le sprite mode en "Multiple" puisque chaque image PNG contenait plusieurs tiles. En ajustant le paramètre "Pixels Per Unit" à 32, j'ai veillé à ce que les tiles aient une taille de 32x32 pixels, correspondant à notre style pixel art.

J'ai également choisi le "Filter Mode" approprié, à savoir "Point", qui est idéal pour préserver les détails pixelisés propres au pixel art, sans ajouter d'effet de lissage indésirable.

Enfin, j'ai utilisé l'éditeur de sprites (Sprite Editor) pour découper les images PNG en tiles individuels. En sélectionnant le mode "Grid By Cell Size" et en réglant la taille des cellules à 32 pixels, les images ont été automatiquement découpées en tiles distincts. Cela nous a facilité la tâche pour les utiliser dans la création des maps.

Grâce à ces étapes de configuration des tiles, nous avons pu préparer les graphismes téléchargés pour une utilisation optimale dans le système de tilemap de Unity. Cela nous a permis de construire rapidement et facilement des maps en agençant les tiles pour créer des environnements cohérents et immersifs dans notre ville post-apocalyptique.

Malgré les difficultés rencontrées au départ, je suis satisfait d'avoir trouvé l'asset adéquat et d'avoir réussi à le configurer pour créer des maps conformes à l'ambiance recherchée. Je suis convaincu que ces maps contribueront à l'expérience de jeu immersive et captivante que nous souhaitons offrir aux joueurs.



Après avoir téléchargé l'asset pour les maps, nous devons effectuer quelques étapes de configuration pour pouvoir utiliser les tiles dans Unity. Lorsque nous récupérons l'asset, il est généralement fourni sous forme d'images PNG, qui doivent être traitées pour être transformées en tiles utilisables.

La première étape consiste à ajuster les paramètres du sprite pour chaque image PNG. Nous devons définir le sprite mode sur "Multiple" car une seule image peut contenir plusieurs tiles. Ensuite, nous réglons le nombre de pixels par unité (pixels per unit) sur 32, car nos tiles ont une taille de 32x32 pixels. Ce réglage assure que les tiles seront affichés correctement à l'échelle dans Unity.

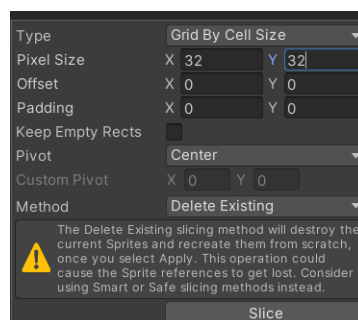
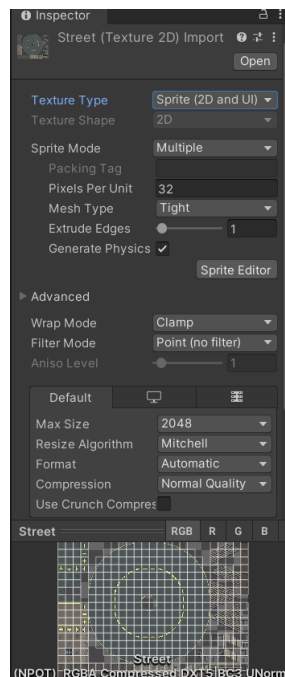
En ce qui concerne le filter mode, nous le configurons sur "Point" pour le pixel art. Cette option évite tout lissage ou flou indésirable et préserve les détails pixelisés caractéristiques du style pixel art.

Une fois que ces paramètres ont été configurés, nous utilisons l'éditeur de sprites (Sprite Editor) intégré à Unity pour découper les images PNG en tiles individuels. Pour ce faire, nous ouvrons l'éditeur de sprites, puis nous nous rendons dans la section "Slice" (découper). Dans cette section, nous choisissons le type de découpe à "Grid by cell size" (découpe en grille par taille de cellule) et nous fixons la taille de la cellule (Pixel Size) à 32. Cette opération permet de découper automatiquement l'image en tiles distincts, en utilisant la taille spécifiée.

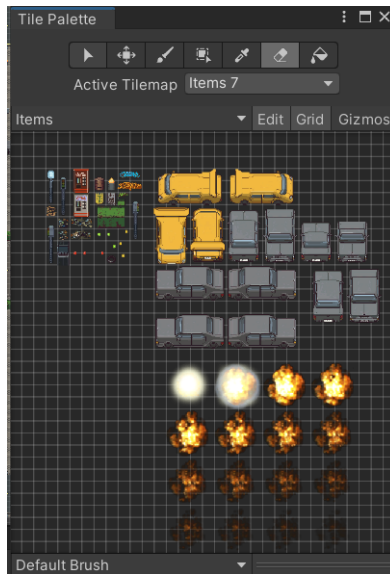
Une fois que les images PNG ont été transformées en tiles individuels grâce à cette découpe, nous sommes prêts à les utiliser pour créer nos maps. Les tiles peuvent être placés et agencés dans le système de tilemap de Unity pour construire les environnements de notre jeu pixel art 2D.

En résumé, la configuration des tiles à partir des images PNG implique de régler le sprite mode sur "Multiple", le nombre de pixels par unité sur 32, et le filter mode sur "Point". Ensuite, nous utilisons l'éditeur de sprites pour découper les images en tiles en utilisant la découpe en grille par taille de cellule, avec une taille de cellule correspondante à la taille des tiles.

Une fois que cette configuration est terminée, nous pouvons utiliser les tiles dans Unity pour créer nos maps, en agencant les tiles pour construire les différents éléments de notre ville post-apocalyptique.



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 réaliser 2 maps pour la première soutenance: 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. Pour la 2eme soutenance j'ai réalisé une 3eme map et pour la dernière soutenance j'ai fait la moitié des autres cartes restantes.

Toutes les maps sont reliées entre elles même si elles sont toutes séparées par des murs invisibles.

10 Ressenti

Ce projet fut une expérience assez spéciale et particulière pour moi, travailler sur un jeu en multijoueur peut être autant amusant qu'éprouvant. J'ai aimé passer du temps à faire les maps, me documenter en regardant des vidéos tutorielles, tester pleins de choses et implémenter les ennemis. Mais lorsqu'il y avait un bug ou plusieurs bugs, cela pouvait parfois prendre des jours pour le réparer et il y a même des bugs que nous n'avons pas pu réparer. Le fait d'être 3 au lieu de 4 personnes a beaucoup joué dans l'avancement du jeu, certaines tâches auraient pu être faites plus rapidement. Nous avons aussi un problème avec git, ce qui faisait que nous ne pouvions pas travailler en même temps, quand une personne travaillait il devait cloner la nouvelle version push sinon il n'aurait pas les nouvelles modifications. Cela affectait un peu le moral, j'avais souvent une baisse de motivation pour avancer dans le projet à cause de des bugs et du problème de git. Finalement, même si tout ce qu'on avait prévu de faire n'a pas été implémenté, je pense que le jeu est jouable et relativement correct vu tous les problèmes rencontrés. Je suis assez content du résultat final.

Part III

Rémi

11 Chronologie de Rémi

11.1 Le Mage (les débuts)

Lors du début du projet, je n'avais jamais travaillé sur unity avant. J'ai décidé de commencer à travailler sur le personnage Mage. Pensant que cela allait être facile. Pour rappel, ce personnage 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.

J'avais donc commencé par implémenter le système de tir. Mais celui-ci a été plus compliqué que prévu à implémenter, du fait que je ne savais pas trop comment utiliser unity. La prise en main et l'implémentation du mage m'ont donc pris pas mal de temps. Mais une fois cela fait j'étais ensuite bien plus à l'aise et rapide sur les autres tâches.

Mais au tout début 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 a ensuite fallu 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.

J'avais donc réussi à en faire une première version mais bien que fonctionnel, elle 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.

11.2 Le site (les débuts)

Je me suis ensuite attaqué au site internet. 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 lancé dans la création du site internet. J'ai créé le site en utilisant

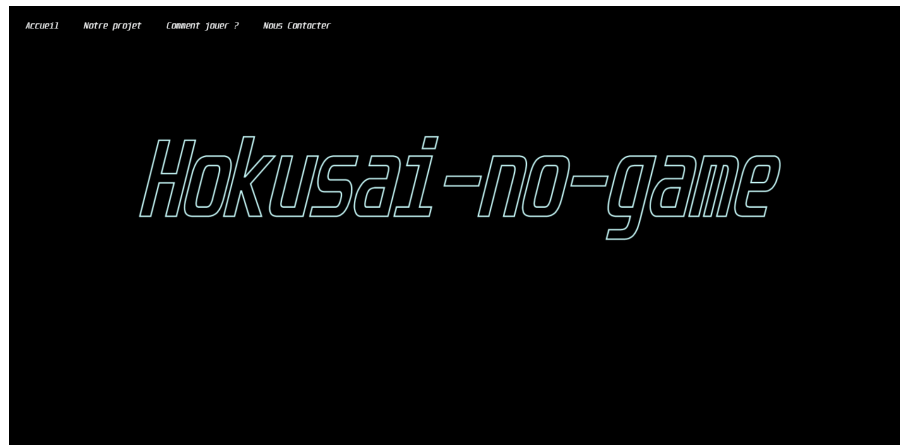
principalement le langage HTML/CSS, ainsi qu'un peu de PHP pour le côté interactif. À noter que son développement est encore en cours et ne sera complètement fini que lorsque le jeu lui-même sera terminé. J'ai quand même créé une grande partie de la base du site.

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 laisser guider par mon inspiration au fur et à mesure de l'avancée du site.

Pour la première 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 début du site :



Ensuite j'ai ajouté le texte, sous forme de petit bloc carré, afin d'apporter de la légèreté au texte et de le rendre plus en adéquation avec le style du jeu.

Image d'une partie de texte :

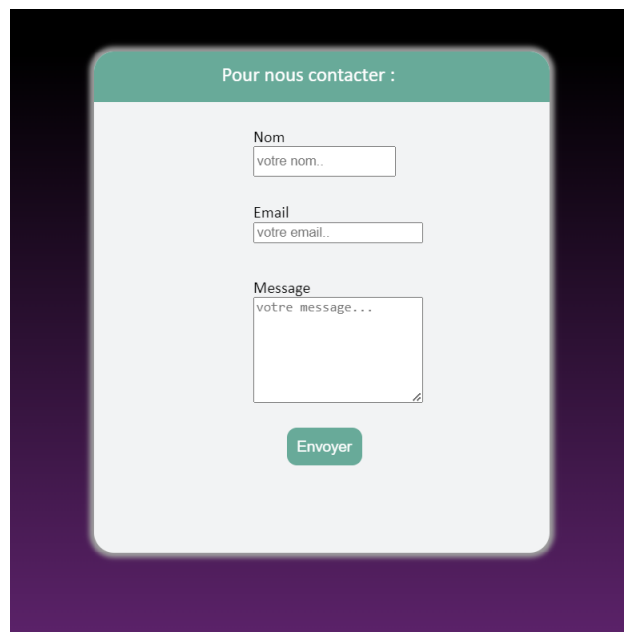


J'ai ensuite, et cela pour toutes les pages, mis une forme traditionnelle de fin de page, dans laquelle on peut nous contacter, ou tout simplement trouver des informations supplémentaires :

Pour les 2eme et 3eme pages, j'ai simplement repris le code de la première en le

réadapant, 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. Pour cela j'ai réutiliser du code que j'avais déjà fait pour un autre site internet. Ce qui m'a permis de sauver beaucoup de temps, car ce système m'avait été compliqué à mettre en place quand je l'avais fait.

Image de la zone de message :

The image shows a contact form titled "Pour nous contacter :". It is set against a dark purple background. The form itself is a light gray rounded rectangle. It contains three input fields: "Nom" with the placeholder text "votre nom..", "Email" with the placeholder text "votre email..", and "Message" with the placeholder text "votre message...". Below these fields is a green button with the text "Envoyer".

11.3 Systeme d'amélioration (les debuts)

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). Puis quand un joueur touche la potion il gagne des points de vie. Comme ce n'était que la 2eme tache apres le mage que j'effectuer, cela m'a pris un peu de temps et j'ai

du bienc omprendre le système distanciation d'objet. J'ai ensuite du reflechir a un moyen qui fait que seul quans un joueur touche la potion celle-ci disparaisse et lui donne de la vie. Pour cela j'ai eu pas mal de difficulter , puis j'ai eu une idée toute simple, utilisé le tag des joueurs. En effet afin de reconetre un jouer, 'lon avait deja uentifier celui ci avec un tag "player". J'ai donc tout simplement utiliser ce tag et le probleme a été vite régler. Au final cette implement de la potion m'a pris un peu de temps et de difficulté mais une fois faite m'a permis d'acquérir beaucoup d'expérience. A noter que un zombie a une chance sur 4 de lacher une potion, histoire de ne pas rendre le jeu trop facile ou trop difficile.

11.4 Le Mage (la suite)

Apres la premiere soutenance, j'ai décidé de contunier sur l'implementation du Mage, Et pour cela je me suis 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 appuyé sur "e". Pour remédier à ce problème, j'ai donc dû donner

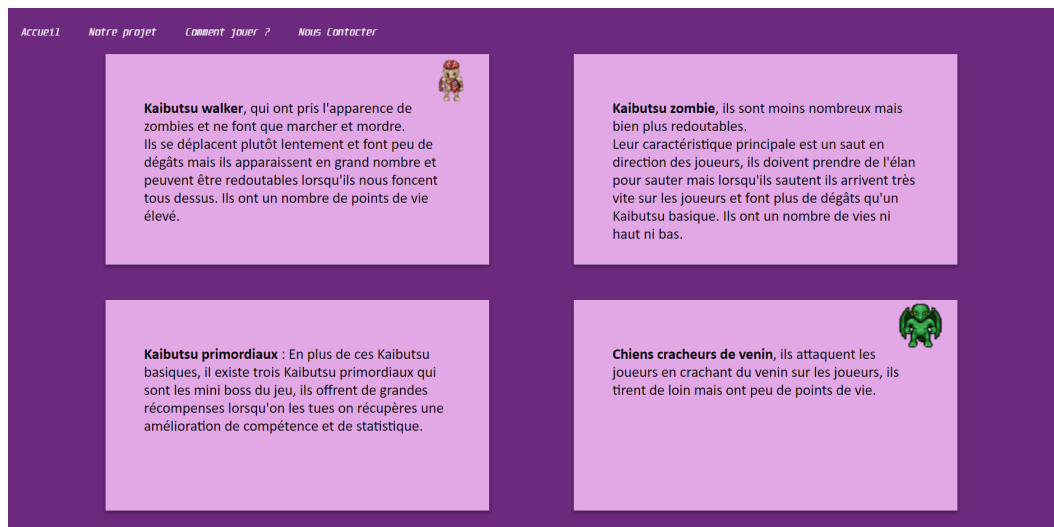
des caractéristiques de déplacement au cercle jaune pour qu'il puisse suivre le joueur et rester tout le temps au centre de lui.

11.5 Le site (la suite)

Après avoir créé la base du site, il me rester encore à ajouté les liens de nos sources comme demandé, c'est-à-dire tous les liens, que ce soit : des membres, des logiciels, des images, des sons, des bibliothèques, des applets et autres éléments que nous avons utilisés. Pour cela j'ai simplement mis des liens qui nous rediriger vers les pages des sites où se trouvent les ressources utilisées.

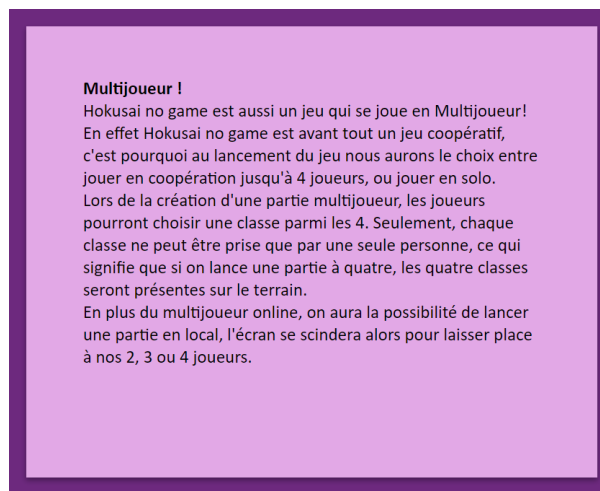


Puis j'ai ajouté plusieurs éléments permettant de compléter la présentation du jeu, notamment sur les différents types de zombie et joueur.

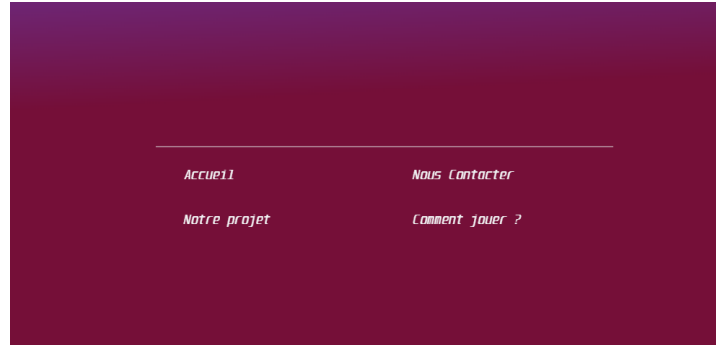


Puis j'ai ensuite rajout quelques petites information et mise en forme :

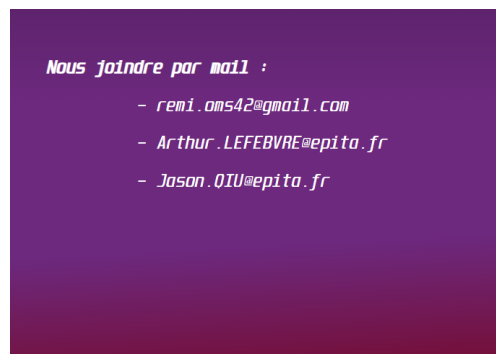
Dans la fin de la presentation du jeu :



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.



11.6 L'assassin

Puis après la 2ème soutenance, j'ai commencé à travailler sur le dernier personnage : 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.

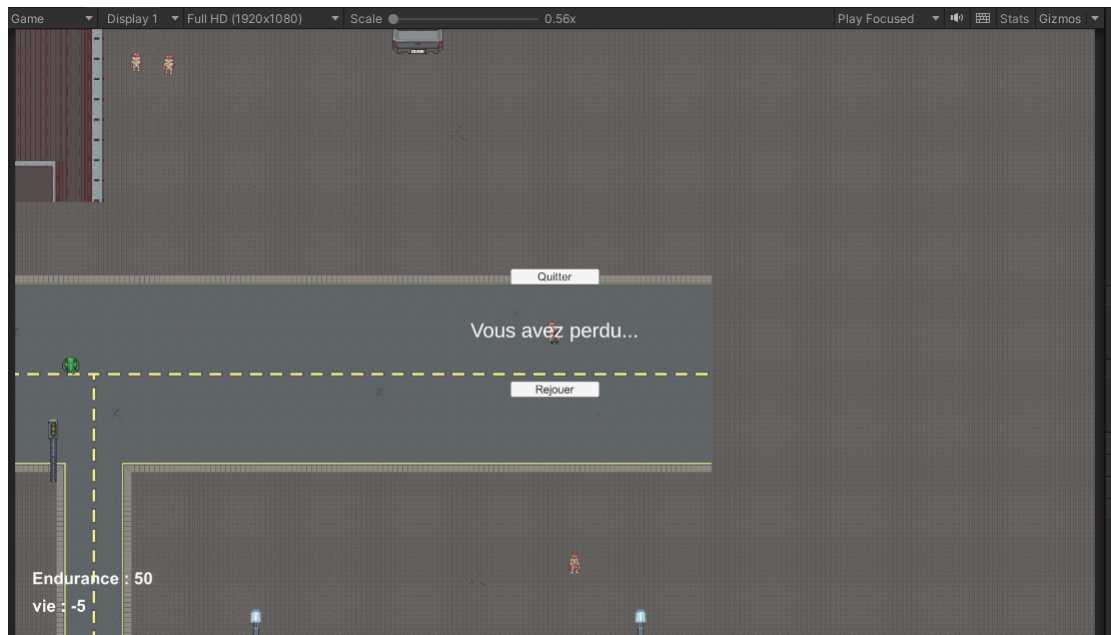
Pour commencer l'implémentation de ce personnage, je me suis inspiré de ce qui avait déjà été fait avant pour le mage et le guerrier afin de gagner du temps. Au niveau de l'attaque, il a presque la même que le guerrier, c'est-à-dire des coups d'épée au corps à corps. Cela n'a donc pas été trop difficile 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 la même apparence que le guerrier.

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.

11.7 Le menu de fin de jeu (perdu)

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.

11.8 Correction de dysfonctionnement

Après la 2eme soutenance, nous avons donc commencer à entamer les derniers ajout. Mais nous nous sommes rendu compte que il y avait quelques bug à régler avant de pouvoir avancer sereinement.

J'ai donc régler différents problèmes tels que :

Un dysfonctionnement au niveau des dégâts : en effet jusqu'à là il n'y avait que le mage qui prenait des dégâts. C'est à dire que peu importe si l'on jouait un guerrier, ou un archer, seul la vie du mage (dans le prefab du player) ne diminuait. Je ne comprenais pas trop d'où venait l'erreur. J'ai donc essayé de comprendre le pourquoi du comment, en changeant pleins de choses et en essayant plusieurs solutions, mais au final rien ne marchait. J'ai donc une idée qui était de pouvoir différencier les personnages afin de savoir à laquelle il faut infliger des dégâts. J'ai donc utilisé une variable "numero-personnage" qui est de 1 pour le guerrier, de 2 pour l'archer etc.. Et grâce à cela j'ai pu adapter le code des ennemis de telle façon à ce qu'ils puissent savoir si ils attaquent un Mage ou un archer.

Un autre dysfonctionnement était que le Mage et l'archer ne tiraient pas à la même vitesse suivant le curseur de la souris. C'est à dire que plus le curseur du joueur était loin du joueur au moment du tir, plus la flèche allait vite. J'ai donc essayé de trouver d'où venait ce comportement indésirable, et je l'ai trouvé. Cela venait tout simplement de la façon donc le système de tir fonctionnait. Pour régler ce problème j'ai donc apporté quelques modifications, qui ont été au final plutôt simples à ma surprise.

Bien sûr il y a eu d'autres petits dysfonctionnements mineurs que nous avons découverts et corrigés tout au long du développement du jeu. Mais ces 2 là étaient les 2 principaux auxquels j'ai été confronté. Après cela j'ai donc pu reprendre la dernière étape de l'implémentation du Mage.

11.9 Le Mage (fin)

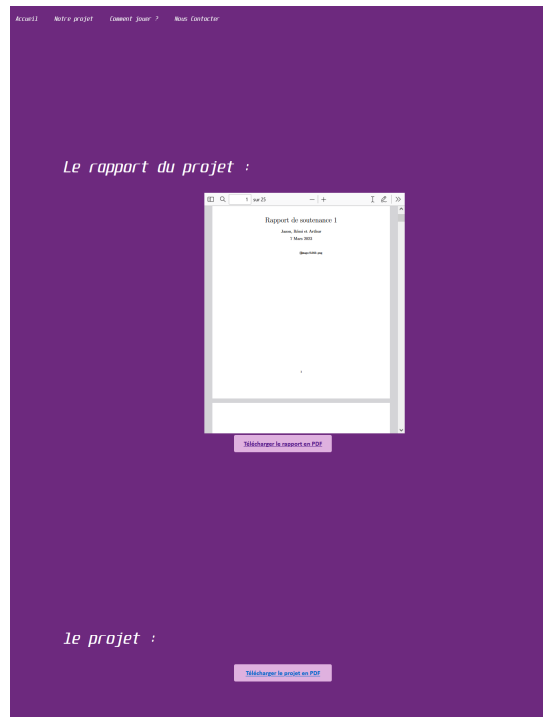
Pour en finir avec le Mage, il fallait donc lui trouver une apparence et des animations unique, car jusqu'a la il avait l'apparence du guerrier. Pour ce faire je me suis inspiré et aider du travaille deja fait pas mes camarade sur le guerrier et l'archer. J'ai donc commencé par chercher un physique qui lui allait bien puis j'ai ajouter ces images dans unity. Et a partir de l'a j'ai donc pu faire des animations (en enchainant plusieurs images). J'ai ensuite relier ces animation au Mage et je les ait ensuite programmer pour qu'elle ne se déclenche qu'a certains moment. Par exemple si le joueur tir en haut, l'animation du tir vers le haut se déclenchera, si le joueur marche vers la gauche, l'animation de marche vers la gauche se déclenchera etc.. Et pour cela j'ai crée un nouveau script qui permet de controler les animation, puis j'ai : d'une part apporter quelques modification sur le code du mage, et d'autre par utiliser les outils de unity pour mieux les controler. Cette partie a été plutôt amusante et m'a permis de connaître les animation sur unity plus en détails Un petit aperçu finale du mage :



11.10 Le site internet(fin)

Pour finire le site, j'ai rajouter les quelques dispositif de fin, tel que : les derniers zombie et leur explication, le rapport de soutenance consultable et telechargeable et enfin, le jeu telechageable. Ces ajouts on été plutot facile dans l'ensemble et ne m'on pas causée beacoup de difficultés.

Illusatration du rapport de projet sur le site :

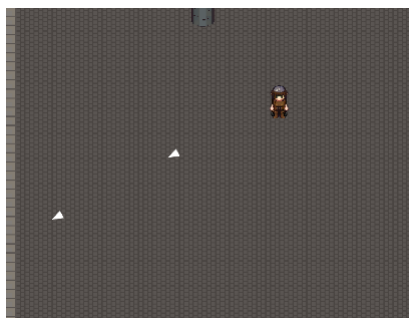


11.11 Systeme d'amélioration (fin)

Enfin la dernière chose sur laquelle j'ai travaillé était le Système d'amélioration. Cela consiste au fait que les personnages et ennemis évoluent en fonction du niveau. C'est à dire que les joueurs, surtout leurs armes, seront améliorés au fur et à mesure

des niveaux. Pour cela j'ai fait plusieurs modifications sur le code des personnages de telle façon à ce que :

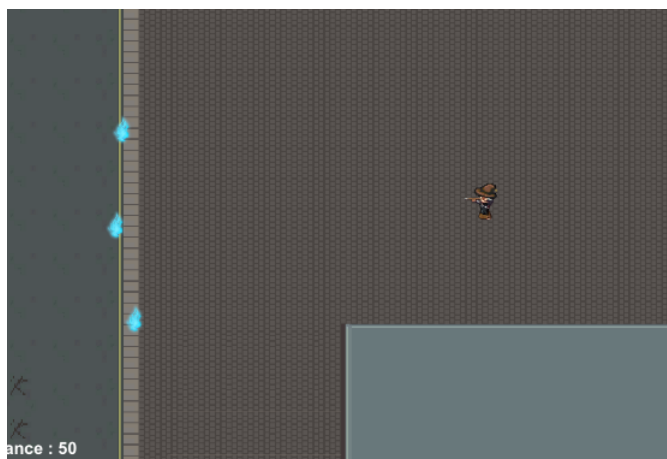
Pour l'Archer : il tire une flèche au début du jeu, puis à partir d'un certain niveau, c'est 2 flèches à 0.2 sec d'intervalle qui sont tirées quand le joueur tire. Puis encore quelques niveaux après c'est 3 flèches à 0.2 sec d'intervalle qui sont tirées quand le joueur tire. Et encore après, les flèches changent et deviennent plus grandes. Ce qui rend l'Archer beaucoup plus agréable et amusant à jouer.



Pour le guerrier, j'ai fait en sorte que au debut, il ne puisse attaquer que toutes les 1 secondes et a une certaine distance de l'ennemies. Puis au fur et a mesure des niveau, il peut attaquer toutes les 0.7 secondes puis toutes les 0.5 secondes. En parallele, sont rayon d'attaque augmente, c'est a dire qu'il pourra faire des degat dans une porté de plus en plus grande.

Pour le Mage, j'ai la aussi fait un systeme ou il peut tiré 1 fois par secondes. Puis avec l'avancer des niveau, il peut tiré toutes les 0.5 puis 0.2 secondes. Et en parallele, a partir d'un certain niveau la aussi, il tire n'on plus 1 fois devant lui, mais desormais 3 fleche, une devant, une legerement sur la droite et une legerement sur la gauche. Ce qui le rend tres fort en fin de jeu.

Image du mage en fin de jeu quand il tire :



bigskip Pour l'Assassin, j'ai fait quasiment comme le guerrier, avec un systeme d'amelioration de l'attaque. Mais en plus de cela j'ai rejouter un systeme d'amelioration sur sa capacité. Au debut du jeu il peut devenir incisible pendant 5 secondes toutes les 10 secondes. Puis a partir plus il avance dans le jeu plus il devien invisible pendant longtemps et plus il peut utiliser sa capciter souvent. Ce qui le rend très efficace en fin de partie.

Pour faire ce systeme de niveau, je ne savais pas trop comment faire au debut, et je pensait que cela llait etres une tache assez compliquer, mais enfaite non. J'ai tous simplement crée une variable "niveau" qui s'incrémente de 1 a chaque fois que le joueur arrive a tué tout les ennemies de la partie dans laquelle il est.

11.12 Mes impresions

J'ai trouvé ce projet assez amusant riche d'enseignement. Bien que la prise en main m'ait pris un peu de temps, j'ai neansmoins appris pas mal de technique sur unity. Le debut été un peu long et fastidue, du fait de ne pas reussir a faire exactement ce que je voulais, ou ne pas comprendre pourquoi tel ou tel chose ne marchait pas. Mais au fur et a mesure de l'avancé du projet j'arrivais

a mieux comprendre ce que je faisant, pourquoi je le faisais, et a mieux et plus rapidement implémenter les taches que j'avais. Et vers la fin , j'aimais bien implémenté les derniere taches et je suis plutot fier de ce que j'au reussia faire. Finalemnt je dirait que la partie que j'ai le moins aimer et ou j'ai le plus eu de

soucis été au debut, avec le mage et la potion de soin des zombie. Et le moment que j'ai le plus aimer été de faire l'animation et le systeme d'ameloration du mage. Car cat partie été plutôt facile et avait un vrai grand impacte dans le jeu. Au final j'ai bien aimé ce projet m'a permet de bien connaitre Unity et les

bases des eju en 2D, et j'en tire donc une expreiences positif.

Part IV

Arhur

12 Caractéristiques des personnages

12.1 Classes des personnages

Les joueurs auront au début de l'aventure le choix entre différentes classes :

- Le guerrier, équipé d'une épée il peut faire de gros dégâts au corps à corps, il peut également parer quelques coups et il a plus de point de vie que les autres.
- L'archer, équipé d'un arc il peut tirer de loin, il y a un système de chargement de l'arc qui fait plus de dégâts une fois complètement chargé, l'archer peut faire un dash.

Chacun des personnages a des compétences en commun : de pouvoir se soigner en récupérant des objets au sol.

12.2 Le guerrier

Pour le contrôle du personnage, les joueurs pourront contrôler l'orientation de leur personnage(c'est-à-dire la direction où ils regardent) en bougeant leur curseur via leur souris. Les déplacements des personnages se feront avec les touches 'z', 'q', 's', 'd', qui seront respectivement se déplacer vers le haut, la gauche, le bas et la droite. Les touches 'e', et 'espace' seront réservées pour les compétences.L'attaque de base sera sur le clique gauche de la souris.

Pour le début du projet j'ai commencé par créer un personnage simple a l'apparence d'un cube qui est capable de faire des mouvement basique dans l'espace. Le personnage est capable de se déplacer dans les 4 direction ainsi que de courir en appuyant sur shift. Pour se faire je récupère grâce à la méthode `unity Input.GetAxis` la direction dans lequel veux se déplacer le joueur puis je crée un vecteur de vitesse du joueur en multipliant l'axe avec une vitesse prédéfini dans une variable privée `speed`. Je modifie cette variable lorsque le joueur appuis sur shift et je rajoute un booleen 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 joueurs 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.

Pour l'attaque, 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. Une 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 ennemis de ce layer, je mets le layer en paramètre de player 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 ennemis dans ce rayon ce qui comprend donc également les ennemis dans le dos de celui-ci. Je récupère donc la direction dans laquelle regarde le joueur à l'aide du curseur de la souris, et je ne récupère alors seulement que les ennemis dans la direction du curseur.

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.

12.3 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.

Pour l'attaque basique de l'archer je dois à chaque update vérifier si le joueur appuie 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. Grâce 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.

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.

Pour la deuxième soutenance je fixe un bug avec le dash de l'archer, qui dorénavant ce fait a l'aide de l'application d'une force en un point donner ce qui créé un dash beaucoup plus agréable a l'oeil et beaucoup plus logique qu'une brève augmentation de la vitesse.

Finalement pour la dernière soutenance j'ai décidé de rajouter un tir alternatif qui permet à l'archer de tirer 9 flèches tout autour de lui, cette capacité le ralentit plus que pour charger un flèche il prend donc plus de risque mais cela lui permet de faire beaucoup plus de mort qui si il ne tire qu'une seul flèche. La capacité est disponible tout le temps lorsque l'on appuie sur espace avec l'archer.

13 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 et 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.

13.1 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 crée 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 multijoueur. Dans la méthode awake je me connecte à photon dans la région 'eu', cette connexion va appeler la méthode OnConnectedToMaster(), dans cette méthode je rejoins le lobby du serveur puis dans la méthode suivante 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 rejoint 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.

13.2 création d'un personnage par joueurs

Pour que chacun ait son propre joueur on doit pouvoir instancier un nouveau joueur à chaque fois que quelqu'un rejoint la partie. Je rajoute donc le pre-

fab du joueur dans les ressource du 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 ajouter 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 créé j'instantie 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 créé mais j'ai plusieurs problèmes tout d'abords c'est la caméra qui est unique aux deux joueurs et ne se déplace pas. Et le deuxième problème et 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 liée, je rajoute donc de 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 joueurs seulement si c'est le siens pour le vérifier il suffit d'utiliser la méthode de `photon gameObject.GetPhotonView().IsMine` qui renvoi un boolean indiquant si le gameobject en paramètre est le siens ou pas. Enfin il me reste le problème de caméra pour commencer je supprime la caméra qui était sur la scène au début. Puis a 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 mets sur orthographique et que l'on positionne au-dessus du joueurs. Ensuite on déplace la caméra en même temps que le joueur mais a une distance fixe sur Z afin de toujours avoir une vue de dessus sur le joueur.

13.3 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 on 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 un autre indiquant si la scène a été load. Dans update si le joueur n'est pas connecter je vérifie si le joueur appuis sur entrée et alors je récupère le code rentrer dans la zone de texte si il fait exactement 4 caractères et le stock 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 un 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 boolean 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 boolean 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.

13.4 Mise en commun des pv des monstres

À la soutenance 1 le multi fonctionnait correctement on arrive à se connecter et à se voir de façon synchronisée par rapport à la position des joueurs sur les différents clients mais un gros problème est soulevé en effet lorsqu'un joueur attaque un monstre lorsqu'il le tue, il n'est pas tué sur l'écran du deuxième joueur et à partir de là les joueurs sont désynchronisés. J'ai pris énormément de temps avant de comprendre d'où venait l'erreur et en fait les points de vie des monstres ainsi que la destruction de ceux-ci se faisaient de la mauvaise façon. Premièrement la suppression des monstres se faisait grâce à la méthode `destroy` de Unity or celle-ci ne fait que détruire le `gameObject` localement et donc si l'on veut le supprimer sur tous les clients interconnectés dans la même partie alors il faut faire appel à une méthode similaire de la bibliothèque `photonnetwork` qui appelle alors grâce à un événement la méthode pour détruire ce `game object` dans tous les clients, alors c'est bon problème réglé ? Et bien non puisque cette méthode ne peut être appelée seulement sur un `gameObject` qui nous appartient, cette appartenance est définie par celui qui a instancié le monstre et on peut la changer également grâce à une méthode de `photon`. Il m'a fallu du temps mais j'ai fini par le comprendre et donc trouver la solution qui est de changer le propriétaire du `gameObject` lorsqu'un joueur tape un monstre pour ce faire je transfère une variable `player` contenant le joueur qui tape lors de l'appel `dealDamage` ce qui me permet de changer dans cette même méthode le "propriétaire" du monstre et de vérifier si il lui reste des hp et ainsi de suite jusqu'à la destruction potentielle du monstre qui n'aura aucun problème car c'est bien le propriétaire qui demande la destruction de l'objet.

Le second problème vient de la synchronisation des hp des monstres, en effet lorsque le joueur fait des dégâts à un monstre il fait baisser le nombre d'hp de celui-ci si or cette valeur est locale et n'est pas synchronisée avec les autres joueurs pour ce faire j'ai utilisé une fonctionnalité de `photon` que je n'avais par encore utilisée qui est la méthode `OnPhotonSerializeView` qui permet à `photon` d'envoyer plusieurs fois par secondes des valeurs et d'en recevoir, je l'utilise pour

envoyer lorsque je suis propriétaires d'un monstre les hp actuel du monstre et lorsque je ne suis pas propriétaire du monstre je reçoit la valeur du propriétaire et je la mets a jour ce qui permet aux deux joueurs d'additionner leurs dégâts si il tapent la même cible.

13.5 Mise en commun de la map et du changment de map selon le propriétaire du serveur

En effet je pense que l'ajout qui a créé le plus de bug au niveau du multijoueur fut l'ajout du changment de maps. Le premier problème fut le déplacement entre les niveaux des différents joueurs car lorsque l'on load un nouveau niveau en multijoueur grace à photon il nous est impossible de conserver les joueurs et leurs valeur ainsi que la camer a et les UI, la seule façon de faire était de recréé un joueur avec les même caractéristique a chaque fois que l'on charge un niveau or si l'on veut éviter à avoir a stoquer des infoirmations qui ne nous seront pas utile pendant le reste du jeu et en plus d'allourdire le changment de niveau la seule solution qu'il nous restent est de charger toute la map au début du jeu et de simplement téléporter les joueurs lors du changement de niveau. Cette façon de faire fut très simple a mettre en place et une fois que l'on avait fait chaque mas individuellement nous les avons reliée s en une seul grande map. Pour téléporter les joueur je vérifie si il reste des ennemies et lorsqu'il n'y en a plus alors grâce a une méthode RPC(de photon qui me permet d'appeler des méthode spécifique à un client avec des information d'un autre client) de téléporter les joueurs au centre du niveau choisis par le gérant de la partie qui aura sélectionner sur la carte le prochain niveau. Une fois teleporter j'instancie les spawnzones qui vont déterminer ou vont apparaître les monstres puis j'attends que les monstres apparaissent pour réactiver la vérification permettant de savoir si les joueurs on gagner le niveau en cours. Si l'on attends pas alors les joueurs peuvent se déplacer sur la carte sans vaincre les monstres car lors du chargement du niveau il y a un petit délai avant l'apparition des premiers monstres.

Pour la synchronisation de la map entre les différents client j'ai utiliser donc une methode RPC mais le prbolème que j'ai rencontrée fut que je ne pouvait pas transférer des tableau de type bool[,] en effet photon est programmer pour ne fonctionner que avec des type simple c'est dire dans notre cas seulement avec des tableau de bool[] simple. Or je n'avait pas envie de perdre beaucoup de temps a devoir changée tout le systeme de changement de niveau et d'update de la carte. J'ai donc créé deux methodes. L'une permet de changée le tableau de bool[,] en tableau de bool[] et l'autre permet de changée le tableau de bool[] en un tableau de bool[,] grâce a une largeur en paramètre. Comme ça je peux transferé le tableau simple et le rechanger un fois réceptionner afin de manipuler les valeurs. Au final cela permet au deux joueurs d'avoir la même crate lorsqu'ils appuient sur m peut importe qu'ils soient le chef de la partie ou non.

14 Animation Archer et Guerrier

14.1 Selection des assets

Pour trouver les assets des personnages nous avons cherché sur internet afin de trouver quelque chose de cohérent avec le reste du jeu tout en correspondant aux personnages que l'on a créé. Pour cela on a fini par trouver un site internet nous permettant de customiser l'apparence d'un personnage et de former automatiquement une feuille d'assets avec dessus des animations du personnage en question de marche, d'action et d'attaque avec ou sans arme. Voici un exemple de feuille d'asset avec une partie de la feuille du guerrier.



Le site en question est <https://sanderfrenken.github.io/Universal-LPC-Spritesheet-Character-Generator>

14.2 Importation dans Unity

Pour importer les animations dans unity je n'ai pas trop eu de problèmes les animations dans la feuille étaient déjà plutôt bien calibrées. J'ai juste eu besoin de faire quelques ajustements au niveau de la vitesse de l'animation qui à l'origine était un petit peu trop rapide.

Une fois les animations importées il faut maintenant que les animations jouent dans les bonnes conditions. Pour commencer j'ai décidé d'appliquer les animations du guerrier à toutes les classes du jeu pour avoir un aperçu des déplacements. Pour ce faire j'ai créé un animator controller qui va permettre de créer des conditions à la lecture des animations. Une fois celui-ci créé je commence par mettre en animation d'origine l'animation de respiration, d'attente qui se jouera par défaut lorsque le joueur ne touche à rien, vu qu'il n'y avait pas d'animation dans la feuille d'asset j'ai simplement sélectionné une image ou le joueur regarde devant lui. Ensuite j'ai ajouté plusieurs variables dans mon animator controller qui seront les deux coordonnées X et Y ce sont deux flottants que je mettrai à jour en fonction des coordonnées de départ du mouvement du joueur et la position du joueur après un petit temps de déplacement, ce qui permet de récupérer la direction dans laquelle il se déplace.

Après avoir ajouter les deux variable je rajoute un arbre d'animation qui se jouera d'origine et qui sera en deux dimension, il jouera selon la valeur de x et y les animation de marche vers la gauche, droite, haut ou vers le bas. J'ai donc remplacer l'animation de respiration par cet arbre, pour que lorsque le joueur ne bouge pas il y est une animation d'attente du joueur j'ai rajouter un booleen dans l'Animator controler que j'activerais dans mes scripts lorsque le joueur n'a pas de mouvement.

Ensuite je rajoute un derniers booleen correspondant a si le joueur attaque ou pas. Lorsque le joueur attaque alors je me déplace dans un autre arbre d'animation qui selon encors une fois les coordonnées x et y jouera l'animation d'attaque dans la bonne direction puis retourneras sur l'arbre de déplacement ou l'animation d'attaque.

Finalement je rajoute les calcul de la direction du joueur dans un script afin d'activer les bonne animation au bon moment en récupérant l'Animator qui est liée au personnage. J'ai ensuite dans le script du guerrier choisi les bon moment pour activer les animation d'attaque afin de correspondre avec le moment ou le joueur clique.

14.3 Animation de l'archer

Une fois que j'ai fini d'implémenter les animations du guerrier, j'ai implémenter les animations de l'archer, je suis donc partie sur une base semblable mais en changeant les animation d'attaque puisque ici le joueur charge un arc il faut donc que l'animation reste en pause a la fin du chargement, j'ai créé un arbre de la même façon que le pour le guerrier mais en changeant le fait que l'animation ne se joue qu'une seule fois et sort de l'arbre seulement grâce a un booleen ce qui fait que maintenant le joueur lorsqu'il attaque charge une attaque en lisant une fois l'animation puis en restant sur la dernière image de l'animation. J'ai ensuite dans le script de l'archer ajouter un moment ou je change le booleen de la fin de chargement pour que lorsque l'on relâche le clique l'animation se débloque.

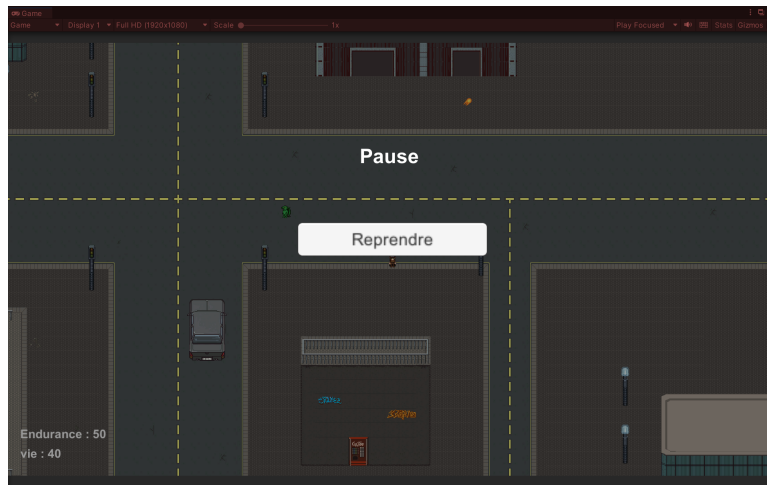
15 Interface

15.1 Lancement d'une partie

Pour que le lancement d'une partie soit plus agréable et que l'on puisse sélectionner une classe j'ai créé une scène spéciale pour la sélection de classe dans laquelle les joueurs peuvent choisir une des trois classes disponible, ces 3 boutons sont liés a un script qui attribue les bon scripts au joueurs pour qu'il est les bonne compétences ainsi que les bonne animations lorsqu'on le sélectionne. Cela créé par contre des problème au niveau du multijoueurs qui ne joue pas les bonne animation pour les autres joueurs et c'est un bug que nous devront corrigée pour la prochaine soutenance.

15.2 Menu de pause

Dans ce menu je vais simplement mettre le jeu en pause seulement si le joueur n'est pas en multijoueur. Pour ce faire je dois commencer par créer un canvas que je vais appeler UI et qui sera à la base de tout les menu dans le jeu. Dans ce canvas je crée un panel qui lui sera dédié au menu de pause je change la couleur en noir et rajoute un bouton qui permettra au joueur de reprendre le cours de sa partie. Un fois cela fait je sauvegarde le prefabrique de ce canvas dans le dossier player pour pouvoir le faire apparaître lors de la création du joueur. Puis dans un script dédié au canvas je mets dans la methode update une vérification afin de savoir si le joueur appuie sur echap qui sera la touche de pause. lorsqu'il appuie sur echap je vérifie grace a un booleen si le jeu est déjà actuellement en pause ou pas. Si le jeu est en pause alors j'appelle la methode afin de quitter la pause et inversement si le jeu n'est pas en pause alors j'appelle la méthode permettant de mettre le jeu en pause. Dans les deux méthodes je mets le temps en pause ou à un cours normal grace a la variable de l'écoulement du temps qui est accessible dans unity grace à `Time.timeScale` qui est une valeur que l'on peut mettre à 0 pour que le temps ne passe pas ou à 1 pour que le temps s'écoule normalement. Je mets actif ou non le panel que je récupère dans une variable afin d'afficher le menu et voila le menu de pause est fini. Il m'a permis par la suite de faire des menu un peu plus complexe. Voici une image d'aperçu du menu simple.



15.3 Menu de selection d'amélioration des statistiques

Pour le deuxième menu j'ai décidé de faire l'amélioration des statistiques de nos héros, en effet cela permet de créer une progression au fur et à mesure de l'avancer dans le monde, cela permet de nous rendre plus fort et de nous

donner le choix de soit finir le jeu vite ou alors amasser des statistique pour nous faciliter la vie mais ce qui nous risque de prendre plus de temps.

Donc pour créé la sélection de statistique j'ai créé un menu simple encore une fois dans un pannel que je désactive par défaut dans ce pannel je mets 3 boutons correspondant aux 3 statistiques et je créé ensuite un script qui récupère le joueur correspondant a l'ui puis je créé un boolean permettant de validée si le joueur a le droit de prendre une statistique ou nous afin d'éviter la triche. Puis je créé 3 méthodes permettant chacune d'améliorer leurs statistique respective la force la vitesse et la vie une fois sélectionner le boolean ce met a false pour ne pas tricher et ajoute les statistiques au joueur, le menu se ferme et le temps reprend. Finalement je fait correspondre l'appui des bouton aux methodes qui leurs corresponds.

Ce menu ne s'affichera que lorsqu'il n'y a plus de zombie sur la carte ce qui signifie que l'on a finie le niveau alors pour ce faire dans la méthode fixeupdate je vérifie si il y a encore des monstres ennemies grâce aux tags ennemies que possèdent chaque ennemies. Une fois cela fait si le nombre d'ennemies est égale à 0 alors j'affiche le menu de sélection de statistiques et je mets le temps à 0. Voici un aperçu du menu de sélection de statistiques



15.4 Menu de la carte et passage au prochain niveau

La carte est composée d'une grille de niveau de 4 par 4 donc pour commencer je créé un pannel toujours dans le même canva dans lequel je mets en forme de grille 16 boutons chacun de ces boutons vont correspondre a un niveau sur la carte. Pour chacun de ces boutons je retire le texte et j'initialise le bouton en haut à gauche comme étant le niveau 1 il est donc en vert afin d'indiquer notre position

Ensuite dans le script du menu je récupère chacun des 16 boutons que je fait correspondre avec des coordonnées dans une matrice, deux matrice pour

être exact une matrice de booleen qui correspond aux niveau qui sont en théorie de niveau coler a nous et sur lesquels on peut se déplacer et enfin un autre matrice de booleen dans laquel on stoque le chemin parcourue par le joueur afin de l'empêcher de faire demi-tour, ensuite je crée une methode pour chaque bouton qui s'active lorsque l'on appuie dessus je vérifie alors si on a le droit de ce déplacer sur ce bouton grace aux deux booleen et puis alors si il a le droit je crée un nouveau niveau je déplace le joueur au milieu et je stock les niveau qui sont à cotées et je mets le niveau dans le chemin des niveau déjà parcouru.

Ensuite je fait correspondre chaque bouton à la méthode qui lui correspond et je crée une méthode permettant de changer la couleur des bouton en vert pour celui sur lesquels on peut se déplacer et rouge pour ceux sur lesquels on s'est déjà déplacé ce qui donne cet aperçu lorsque l'on peut se déplacer:



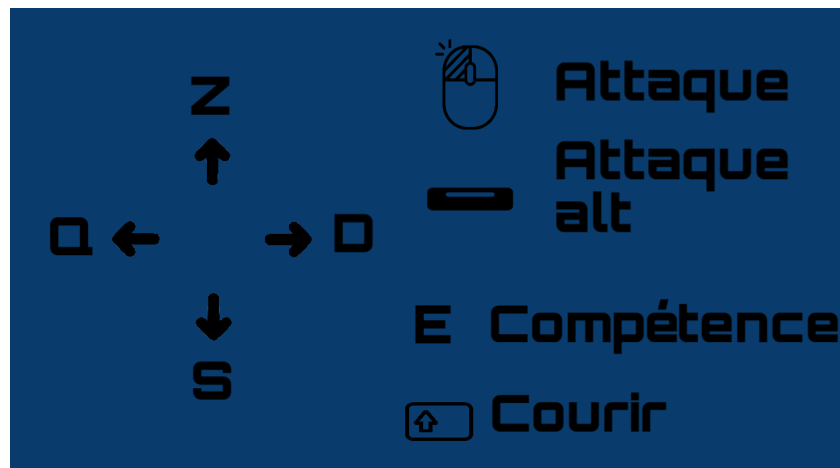
Voici le résultat de l'affichage de la carte lorsque l'on a choisi par exemple de ce déplacer vers la droite on peut remarquer que l'on ne peut pas se déplacer vers la première case ou vers une case en diagonale.



L'ouverture de la map en multi a pauser beaucoup de problème donc comme je l'ai dit plutôt, j'ai synchroniser la carte a chaque fois que la carte change grace aux methode RPC de photon. Et je ne permet seulement que au chef de la partie de pouvoir sélectionner le prochain niveau pour éviter les conflits entre les différents clients.

15.5 Menu de tutoriel

Afin que le joueur ne soit pas perdu lorsque l'on lance une partie je me suis dit que faire apparaître un menu qui prend toute la place de l'écran avec les informations essentiel pour apprendre en peu de temps les touches au nouveau joueur pouvait être une bonne idée car avant nous lancions une partie sans vraiment avoir les touche indiquer quelque part. La page de tutoriel ressemble donc à ça



Il y a tout les contrôles principaux, ce qui permet une compréhension efficace sans pour autant dévoiler tout le jeu ce qui est parfait pour un nouveau joueur.

Si lors du démarrage le joueur appuis sur échap trop tôt ou qu'il veut révérifier un contrôle il peut re avoir acces à ce menu de tutoriel dans le menu pause accessible en appuyant sur échap une nouvelle fois. Il est important de noter que le jeu seras en pause lors du premier démarage et que donc le joueur a tout son temps pour prendre en compte les différentes touches de contrôle du personnage.

16 Sound effect

Après l'ajout des animations le jeu à déjà plus de charme mais pour le rendre meilleur il nous fallait des bruit et musique anbiante. Pour ce faire rien de plus simple nous utilison les outils intégrer à unity en rajoutant un audio listener sur les caméra pour que chaque joueur et les bruit ainsi que plusieurs sources audio qui émettent des bruit, un dans le niveau gère la musique et d'autre sur les joueurs gère les bruit créé par les compétences par exemple.

Pour pouvoir lire un bruit lors d'une action il suffit d'appeler la bonne source audio avec la méthode play qui permet de lire une fois la source audio.

Pour la musique on peut faire en sorte que tout le monde l'entende peut importe la distance de la source audio qui se trouve au centre de la carte.

17 L'installateur et le désinstallateur

Pour pouvoir faire l'installateur j'ai fait quelques recherches et après avoir naviguer sur différents blog j'en ai conclu que inno était un logiciel totalement approprier pour ce que j'avais à faire.

J'ai lancé Inno Setup et utilisé l'assistant pour commencer à créer le script d'installation, en spécifiant le nom de l'application, le nom de la société et la version de l'application. Ensuite, j'ai défini le dossier de sortie (où l'installateur sera créé) et le dossier d'application par défaut (où le jeu sera installé sur l'ordinateur de l'utilisateur). Dans l'assistant Inno Setup, j'ai ajouté tous les fichiers du jeu au script. Cela comprend le fichier .exe de notre jeu ainsi que tout les fichiers de données associés. Ce qui est bien avec Inno Setup c'est qu'il crée automatiquement un désinstallateur lors de la création de l'installateur. Cela signifie que les utilisateurs peuvent facilement désinstaller le jeu à partir du Panneau de configuration ou du menu Démarrer. Après avoir configuré toutes les options et ajouté tous les fichiers nécessaires, le script c'est compilé et j'avais créer l'installateur totalement fonctionnel.

18 Mes impressions

Travailler sur ce projet a été une sacrée aventure, avec son lot de bons moments et de galères. Au début, j'étais super excité à l'idée d'apprendre à utiliser Unity et C, deux outils que je ne connaissais pas bien.

Mais le départ de deux de nos camarades a compliqué les choses. On s'est retrouvé à devoir gérer plus de travail que prévu. C'était dur, mais on a fait face.

Finalement, malgré les obstacles, j'ai beaucoup appris. J'ai bien progressé en C et Unity, et j'ai découvert LaTeX pour la création de PDF. C'était pas toujours facile, mais je ne regrette rien. C'est sûr, cette expérience m'a beaucoup apporté et le résultat final me plaît malgré le fait qu'il ne soit clairement pas parfait et qu'il manque de contenu.