Rapport de soutenance 2 EPITA - *Projet S2*

HERMITA *Produit par Epistars™*

Anouar BELMDEJENNEH Antoine BLUMENROEDER, Baptiste DURRINGER, Dylan DE ARAUJO *2022-2023*

Table des matières

1 Introduction

1.1 Attendus

Anouar	Dylan	Baptiste	Antoine
Gestion des déplacements	Créations de sorts	Combat en multijoueur	Héberger une partie
Lancer des sorts	Fabrication de l'équipement	Mécaniques des boss	Rejoindre une partie
Interface	Élaborations des potions	Intégration de l'établi	Apparition des ennemis
Stockage Inventaire	Personnage	Agencement du monde	Intelligence artificielle
Enregistrement des statistiques	Ennemis	Établissement du housing	Inventaire fonctionnel
Sauvegarder île de départ	Environnement	Système de combat	Site web

Tâche :
Pas commencée | Incomplète | En retard | Presque finie | Tâche remplie | Retardée par une autre

FIGURE 1 – Réalisations des tâches : Soutenance 2

Pour cette deuxième soutenance, l'objectif était d'obtenir un jeu jouable avec des monstres que l'on pourrait attaquer et qu'ils nous attaquent en retour. Dans l'idée on aurait pu sauvegarder une certaine progression dans le jeu. Nous avons fait face à un certain manque de motivation de la part du chef de projet qui devrait avoir pour mission de donner les directions au projet mais cela n'a pas été le cas. Pour réagir face à cela, les membres du groupe ont pris des initiatives et le projet a tout de même eu l'occasion de bien avancer. Nous avons aussi pris rendez-vous avec le coach pour savoir comment mieux gérer cette situation. Dans la globalité, le projet a pris beaucoup de retard sur les sauvegardes. Nous n'avons toujours aucun sort fonctionnel ce qui a retardé l'implémentation du *craft* des sorts puisque seul un prototype a été montré sans réel sort que l'on pourrait *crafter*. Aucun début d'interface de combat n'est implémenté dans la scène du jeu mais ceci reste un retard rattrapable. Pour ce qui est des déplacements, il était attendu à la première soutenance une gestion des collisions mais un autre membre qui en avait besoin a dû l'implémenter lui-même. Au niveau des graphismes, rien n'est inquiétant, seuls quelques *assets* de l'environnement sont à réaliser pour finaliser la carte. Et finalement, le game desgin est réalisé dans la globalité mais la taille de la carte a entraîné quelques retards sur l'implémentation des détails.

2 Dylan

2.1 Graphisme

2.1.1 Le mage à l'attaque

Dans la continuité de ce qui avait déjà été réalisé, j'ai désormais dessiné les attaques du mages. Une chose était différente dans la réalisations des nouvelles animation : la rapidité du mouvement. Ainsi j'ai appris comment bien placer les différentes trainées derrière les objets qui bougent rapidement, comme dans la Frigure 2 Aussi, j'ai apporté au niveau de la boule au

FIGURE 2 – Attaque 1 du mage

FIGURE 3 – Attaque 2

bout de la canne tenue par le mage un effet de lumière au moment de l'attaque, comme nous pouvons le voir dans la Figure 3

Ainsi j'ai réalisé deux types d'attaques, sur les huit côtés que l'on peut voir dans les Figure **??** et Figure **??**

2.1.2 Attention aux gobelins !

Tout comme le mage, l'asset du gobelin afin qu'il puisse courir était déjà réalisé pour la première soutenance, il manquait plus qu'à réaliser les dessins afin qu'il puisse attaquer en jeu. Le procédé est exactement le même que pour le mage, avec les effets de vitesses apportés au mouvement. Le résultat est visible dans la Figure 6

2.1.3 Un nouvel ennemi en approche : le robot

Afin d'avoir un nouvel ennemie dans le jeu, j'ai décidé de réaliser un robot comme on l'avait prévu avec l'équipe. Les mêmes étapes de création que pour les précédents personnages ont été mises en place afin de mener à bien la réalisation du robot. Pour réaliser ce robot, je me suis inspiré des robots présents dans le film d'animation intitulé « Le Château dans le Ciel » et des golems de fer présents dans Minecraft. Les déplacements de cet ennemi est assez original comparé au gobelin et au mage, puisqu'il ne se déplace pas avec des jambes mais une roue. J'ai

FIGURE 4 – Attaque 1 du mage

FIGURE 5 – Attaque 2 du mage

ressenti une certaine satisfaction en réalisant se déplacement, puisque le dessin et l'animation de la roue n'est pas si compliqué à mettre en place lorsque l'on a bien compris quelle forme doit avoir la roue en fonction du sens du mouvement. L'asset résultante est visible dans la Figure 8

En ce qui concerne l'attaque effectué par cette ennemi, j'ai imaginé que celle-ci pourrait directement venir de sa main. Celle-ci est alors propulsé vers le mage à attaqué grâce à un ressort intégré dans le bras. L'asset obtenue de l'attaque du robot est trouvée dans la Figure 9

2.1.4 Encore un gobelin !

Avec l'équipe, nous avons prévu de réaliser au moins trois ennemis principaux, c'est-à-dire un gobelin, un robot, et un gollem qui arrivera à la prochaine soutenance. Mais nous avons aussi dans l'idée que ces trois types d'ennemis soit en réalité les racines de trois familles d'ennemies. Pour être plus clair, nous avons déjà un gobelin prêt à partir au combat avec sa petite hache dorée. Mais j'ai ici créé un autre gobelin portant une arme différente, une épée. Nous pourrions presque dire que ces deux gobelins se ressemblent comme deux gouttes d'eau, mais celui avec l'épée effectuera plus de dégâts lors de son attaque. C'est ainsi l'idée portée derrière les trois familles d'ennemies ! Pour le gobelin j'ai apporté l'idée de changement d'arme pour montrer la différence de dégâts d'attaque, mais cela pourrait aussi être une différence de points de vie, un déplacement plus rapide, et ainsi de suite ... Les déplacements du nouveau gobelin dont il est question se voient dans la Figure 7

2.2 Système de craft (L'atelier)

L'atelier doit permettre la création des potions et des différentes pièces d'équipement. L'atelier, lors de son ouverture, ressemble à la Figure 10

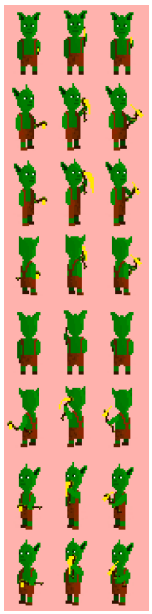


FIGURE 6 – Attaque du gobelin

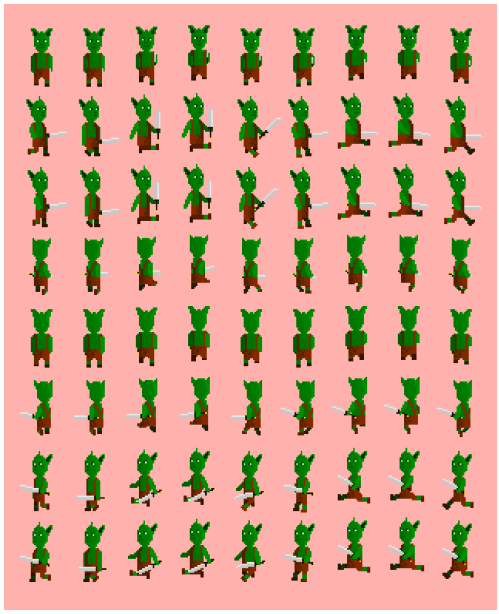


FIGURE 7 – déplacements du gobelin épéiste

2.2.1 Le choix de l'item à créer

A gauche, le joueur peut retrouver l'ensemble des items qu'il est possible de créer grâce à l'atelier. Chaque bouton est une prefab composée de l'image de l'item à fabriquer, elle-même associée au nom de l'item. De plus, chaque bouton associé à son item permet de montrer les éléments nécessaire à la création de ce-dernier comme on peut le voir dans la Figure 11

2.2.2 La recette afin de créer l'item

Au moment du clique sur le bouton pour choisir l'item à fabriquer, ce bouton étant associer à l'identifiant de l'item, il est assez aisée à travers le code de retrouver la recette de celui-ci dans la base de données. C'est cette recette qui est affiché à droite de l'écran. Nous avons ici aussi affaire à des prefabs pour chaque item, composé de l'image de l'item, de son nom, ainsi que la quantité de celui-ci que peut fournir le joueur, comparé à la quantité nécessaire. Jusqu'à 6 items de types différents peuvent être nécessaires lors de la création. Ainsi, pour que le tout reste joli à regardé, j'ai utilisé un composant dans le panel de la recette appelé « Grid Layout Group » afin que, lorsque le nombre d'items dépasse le nombre de trois sur une ligne, les autres soient affichés sur une autre ligne en-dessous.

2.3 Le bouton « craft »

Celui-ci a pour but, comme son nom l'indique, de créer l'item sélectionné. Même s'il est présent, il n'est pas encore fonctionnel. Celui-ci sera associé plus tard à une fonction qui vérifiera que tous les items nécessaires à la fabrication sont bien présents dans l'inventaire, et uniquement si tel est le cas, retirera les éléments utilisés par la recette de l'inventaire et ajoutera l'item créer dans l'inventaire.

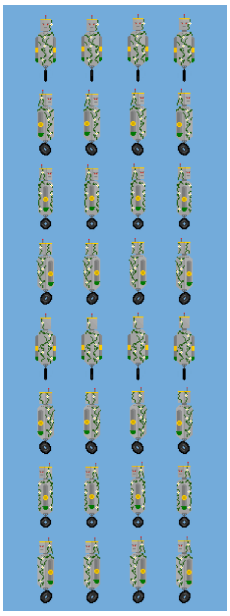


FIGURE 8 – Robot

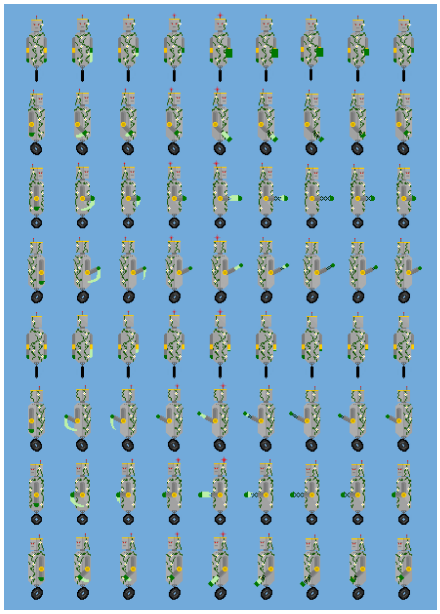


FIGURE 9 – Attaque du robot

FIGURE 10 – Craft à l'ouverture

2.4 Prochaine soutenance

En ce qui concerne le système de craft, l'atelier sera terminé (pour être plus précis, le bouton de craft sera fonctionnel), et le laboratoire auquel je ne me suis pas encore attelé sera lui aussi totalement fonctionnel. Au niveau des graphismes, les trois grandes familles d'ennemies seront entièrement réalisées. Par la suite les dessins de l'établi, l'atelier, et enfin, du laboratoire seront effectués et ajouté au jeu. Différentes assets pour le sol seront réalisées en fonction des demandes de Baptiste pour qu'il puisse avancer sereinement dans le game design. De plus, nous avons discuté avec Baptiste du fait que ce serait bien d'ajouter de la profondeur en dessous de l'île, avec l'apparition de nuages et la présence de terre en dessous de l'île comme dans la Figure 12. Cette dernière tâche sera réalisée uniquement si les autres sont préalablement terminées.

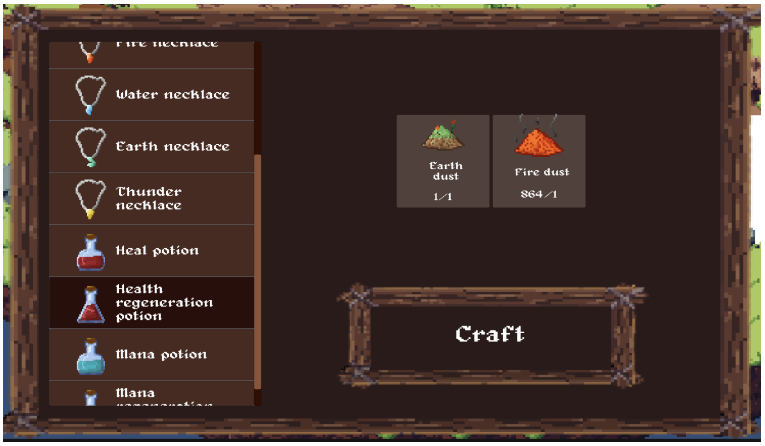


FIGURE 11 – Apparition de la recette dans le craft

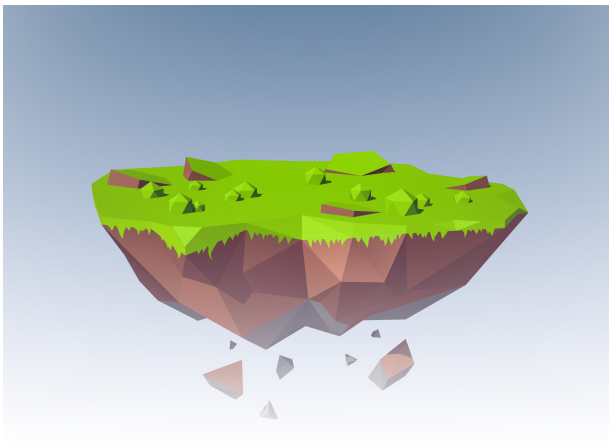


FIGURE 12 – CHANGER

3 Baptiste

3.1 Workbench

Une de mes tâches était de réaliser la workbench, donc la table qui permet de broyer des géodes pour obtenir une gemme aléatoire, de transformer des gemmes brutes en gemmes polies et de transformer les différents éléments. Tous ces crafts se font avec une durée dans différents slots de craft communs. (Figure 13)

3.1.1 Bâtiments

La première étape dans la conception de la workbench a été d'implémenter le bâtiment dans la scène, cela m'a été rendu très facile par le script Building d'Antoine.

J'ai donc implémenté pour chaque bâtiment un simple sprite invisible (qui sera remplacé plus tard par le sprite correspondant au bâtiment, excepté pour le portail qui a deja son sprite animé) avec le script Building associé, ce qui permet au joueur, lorsqu'il est près d'un bâtiment, d'intéragir avec celui-ci et donc d'ouvrir les interfaces associées.

FIGURE 13 – Interface de la workbench

3.1.2 Interface

Pour l'interface, les différents critères étaient qu'il y ait une partie pour broyer des géodes contenues dans l'inventaire, une partie pour sélectionner un craft à effectuer ainsi qu'un affichage du temps restant.

Comme vous pouvez le voir dans la Figure 13 j'ai opté pour un affichage comportant une partie craft à gauche, une partie géode à droite et une partie avec des barres de chargement ainsi que des slots montrant les items qui sont entrain d'être craft.

Pour la partie de gauche j'ai créé une scroll view qui grâce au script de l'objet de l'interface entière se remplie de préfabs de craft adaptés à chaque recettes de la RecipeDatabase ayant l'attribut `Building = "wb X"`. Cela rend la chose très modulaire car pour ajouter une recette, il suffit simplement de l'ajouter à la database. Ensuite, pour l'affichage de l'item sélectionné, tout d'abord, un script associé à chaque préfab de la liste de craft détecte si le joueur le clique, et si oui modifie la variable `_selectedRecipe` avec la recette correspondante au préfab. Grâce à cela, j'ai intégré un panel, en bas à gauche de l'affichage, contenant le nom de l'objet sélectionné, son image ainsi qu'un autre panel contenant ses ingrédients (ici encore une fois j'ai créé des instances d'un préfab `WorkbenchIngredient`) et également le bouton qui permet de lancer le script de craft. (Figure 14)

Puis, pour la partie des géodes, j'ai simplement affiché le nombre de géodes dans l'inventaire du joueur grâce à la fonction `Get_Amount()` de l'inventaire, ainsi qu'un bouton permettant d'effectuer le script de craft de géode.

Pour finir, la partie timers, j'ai créé 3 sliders ainsi que 3 images de slot ayant comme enfant une image de sprite vide qui sera remplacée par le script. (Figure 15)

3.1.3 Script de craft

La première tâche de cette partie a été de remplir la base de données des recettes, pour cela, tout comme la classe `Item`, Antoine avait deja préparé la classe recipe donc j'ai juste eu à la remplir grâce à la liste de craft que j'avais fait au préalable.

L'étape suivante a été d'écrire la coroutine `CraftProgress` qui sera utilisée par le craft de géode et d'objets. Cette fonction actualise WorkbenchUI, puis attend le temps correspondant au temps de craft de l'item sélectionné tout en actualisant la barre de chargement associée. Une fois le temps attendu atteint, si l'objet du craft était une géode une gemme aléatoire est donnée au joeur sinon l'item correspondant au craft lancé lui est donné. Pour finir, la fonction reset toutes les valeurs et objets à leurs états initiaux.

FIGURE 14 – Partie crafting

Ensuite j'ai implémenté la fonction associé au bouton de broyage de géode, celle-ci (dans le cas où un slot de craft est vide et qu'au moins une géode est présente dans l'inventaire du joueur) enlève une géode, lance la coroutine `CraftProgress` et remplace le sprite du slot par une géode.

Pour finir, la fonction `Craft` sert aux crafts des items de la liste sur la gauche. Celle-ci vérifie si un slot de craft est disponible et si le joueur possède les ingrédients nécessaire au craft sélectionné, ensuite tout comme pour les géodes, elle retire les ingrédients de l'inventaire, lance la coroutine et associe le bon sprite d'item au slot de craft.

3.2 Boss fights

Une autre tâche qui m'était donnée était de créer entièrement les combats de boss, j'ai donc commencé par établir la liste des choses que cela comprennait ce qui me donne : créer le boss, gérer ses attaques, gérer son apparition et gérer les salles des combats. Pour cette seconde soutenance, je me suis concentré sur la création du premier boss et sur la création de ses attaques. La gestion de leurs apparitions et des salles de combat se fera en parallèle d'autres éléments du jeu tel que le level design.

3.2.1 Boss

Pour créer chaque boss, j'ai tout d'abord créé une classe `Boss` qui sera associé à chacun d'entre eux, de la même manière que la classe `Monster` hérite de la classe `Entity`. Les boss utilise le même pathfinding que celui des monstres faits par Antoine, leur fonctionnement diffère dans la manière d'attaquer : le monstre a seulement pour rôle d'effectuer des dégâts au joueur lorsqu'il est assez proche. Cependant, pour les boss, nous souhaitions créer des combats plus originaux avec des attaques principalement basées sur des effets de zone à esquiver. Pour une création de boss simplifiée par la suite j'ai voulu encore une fois créer quelque chose de très simple de modification.

FIGURE 15 – Partie géode et slots de craft

3.2.2 Fonctionnement des attaques

Le script `Boss` fonctionne donc de la manière suivante : une liste `attackPrefabs` lui est associée (Figure 17), ce qui permet par la suite directement depuis unity, d'ajouter des prefabs d'attaque dans celle-ci simplement en effectuant un drag and drop. Ces prefabs d'attaque sont également simple à créer (Figure 16), ils sont composé d'un objet `Pivot` qui sert à indiquer où se place le boss par rapport à l'attaque et un objet `AttackZone` qui permet de definir la zone dans laquelle le joueur prendra des dégâts grâce à un `Box Collider 2D`.

Ensuite, pour lancer l'attaque le boss vérifie constamment si le joueur ciblé est dans sa zone de rayon `Range`, si le joueur y est, le boss prend un prefab d'attaque aléatoire puis calcule l'orientation dans laquelle le prefab devra être posé. Pour cela, en vue isométrique, il suffit de mettre l'orientation du X à 65 et du Y à 0 pour avoir un sprite qui donne l'impression d'être posé au sol (Figure 18), il suffit ensuite de changer son orientation Z pour donner l'impression de le faire tourner sur le sol. Pour calculer l'orientation du Z, il faut voir la différence de position entre le boss et le joueur comme un triangle rectangle, ensuite à l'aide de la trigonométrie on peut trouver l'angle recherché en prenant $\arctan(\frac{Ytarget-Yboss}{Xtarget-Xboss})$. Il ne reste donc plus qu'à instancier le prefab au coordonnées de l'enfant `Pivot` du boss qui sert à faire apparaître les sorts à ses pieds et avec l'orientation (65, 0, $\arctan(\frac{Ytarget-Yboss}{Xtarget-Xboss})$).

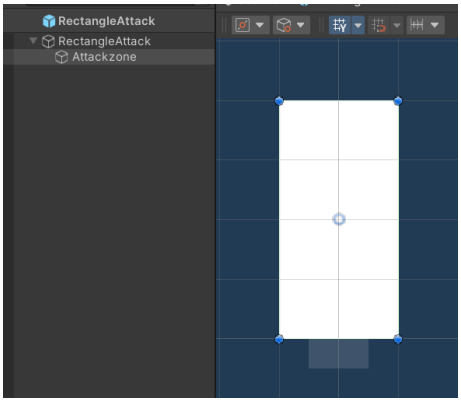


FIGURE 16 – Prefab de l'attaque rectangle

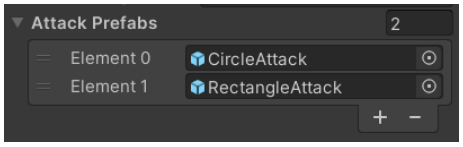


FIGURE 17 – Liste d'attaques du boss à remplir

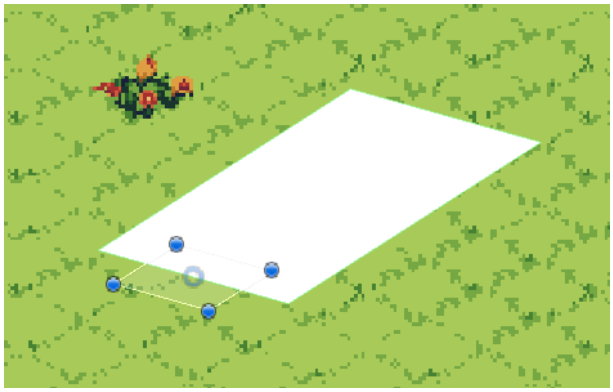


FIGURE 18 – Illusion de sprite posé au sol avec le bon angle

3.2.3 Boss attacks

Ensuite, ces attaques doivent pouvoir faire des dégâts aux joueurs, pour cela, j'ai associé un script aux préfabs d'attaques. Celui-ci a seulement 2 fonctions : la première est la fonction `OnTriggerEnter2D` qui applique les dégâts associés au prefab aux joueurs touchés par le collider grâce à la fonction `TakeDamage(attackDamage)`, sa deuxième fonction est simplement de détruire le prefab après un temps donné (pour l'instant 0.3 seconde) pour cela j'ai utilisé une coroutine.

Dans le futur j'aimerais implémenter le fait que le prefab n'effectue pas directement les dégâts pour que le joueur puisse l'esquiver, avec une animation propre je pense que ca rendra mieux. (Figure 19)

3.2.4 Boss implémentés

Pour le moment j'ai simplement implémenté le premier boss qui sera un roi gobelin, il a pour l'instant la texture du gobelin classique car je ne l'ai pas encore dessiné, tandis que ses attaques sont pour l'instant une attaque devant lui en rectangle et un cercle autour de lui, celles-ci sont clairement sujet à changement mais comme dit précédemment, la manière avec laquelle les boss ont été faits rend l'élaboration de ceux-ci beaucoup plus rapide, simple et agréable. (Figure 20 et Figure 21)

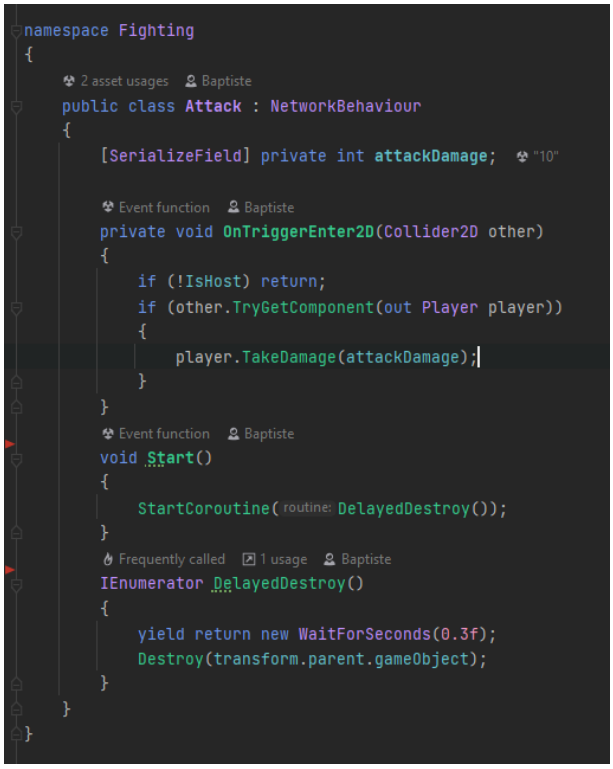


FIGURE 19 – Script du sprite d'attaque du boss

3.3 Level Design

La dernière tâche qui m'a été attribuée était le level design du jeu, cette tâche est celle dans laquelle j'ai le plus de retard, principalement dû au fait que le plan de map initial que j'avais prévu est très grand et qu'il faut à chaque fois dessiner de nouveaux assets car chaque zone est différente. Mais je ne me fais pas de soucis car du fait que j'ai pris de l'avance dans mes autres tâches je vais pouvoir plus m'y consacrer d'ici la fin du projet.

3.3.1 Ajout des lairs

La nouveauté principale par rapport à la précédente soutenance a été l'implémentation des lairs, ceux-ci permettent de faire apparaitre un mob choisit à l'endroit où il est placé. Cet ajout fait par Antoine m'aide beaucoup et va beaucoup m'aider dans le placement des monstres sur la map car il a rendu cet outil très simple d'utilisation. Pour l'instant je n'en ai que placé sur l'île de départ car les seuls mobs que nous avons créé sont ceux qui seront sur cette île. Mais lorsque nous aurons implémenté tous les autres monstres, je pourrai placer tous les mobs de la map. (Figure 22)

3.3.2 Zone de départ

J'ai décidé de revoir totalement l'aspect du temple de départ car le fait qu'il était entièrement en pierre créait un effet pas réaliste et pas très beau visuellement, j'ai donc décidé de partir sur un style de jardins à étages mais toujours sous forme de temple. De cette manière, la zone de départ est beaucoup plus verte et colorée ce qui ressemble plus à une zone de départ qu'un bloc

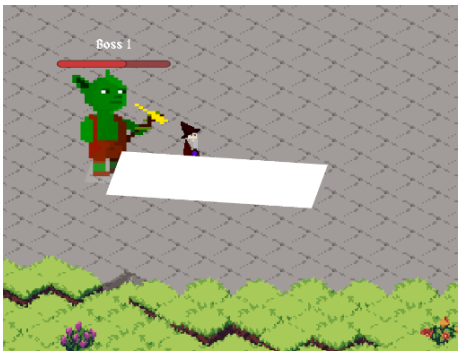


FIGURE 20 – Attaque rectangle du boss

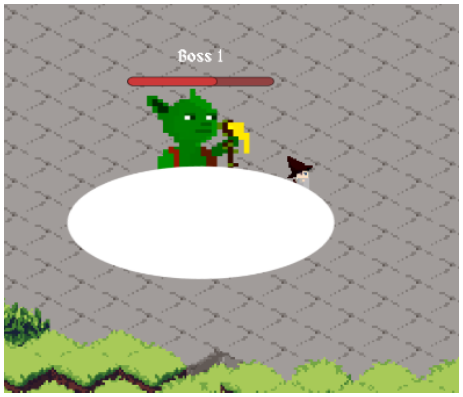


FIGURE 21 – Attaque cercle du boss

FIGURE 22 – Groupe de gobelins placés grâce au système de lair

gris. J'ai également placé notre portail à l'endroit où le joueur apparaît pour plus de cohérence, créé des chemins pour que le joueur se sente guidé, du relief, des fleurs et la zone où le combat du boss final se déroulera. (Figure 23)

3.3.3 Zone ruines

J'ai également commencé la création de la seconde zone : la zone de ruines, j'ai commencé par mettre du relief simplement en rajoutant des couches d'herbe à des endroits pour pas que l'on ait l'impression d'être en permanence sur une plateforme plate. J'ai également fait le tracé des chemins de la zone qui guideront le joueur à travers la map, en plus d'ajouter à leurs côtés les emplacements de certaines des prochaines ruines. J'ai également tenté un essai de ruine de maison, comme nous n'avons pas le temps de dessiner énormement d'assets de débris, de ruines etc... J'ai essayé de me débrouiller simplement avec les assets de terrain que j'avais fait. (Figure 24 et Figure 25)

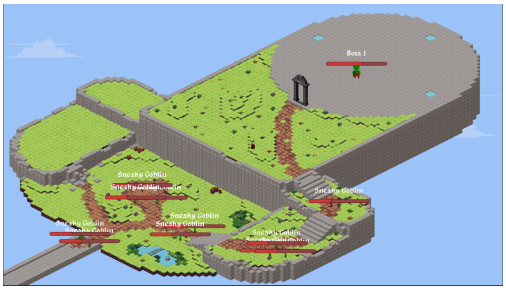


FIGURE 23 – Zone d'arrivée des joueurs

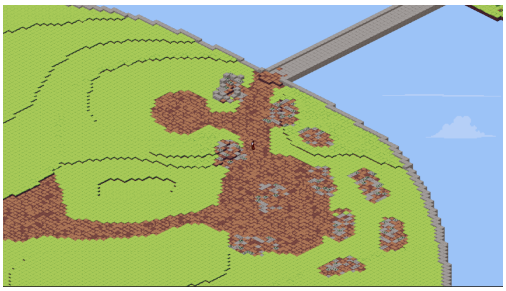


FIGURE 24 – Test de ruine de bâtiment

FIGURE 25 – Test de ruine de bâtiment

3.4 Prochaine soutenance

Et voilà qui conclue cette deuxième soutenance, le programme des choses à faire d'ici la prochaine est plutôt simple, il faut finir le jeu. De mon côté, je dois finir totalement les combats des boss et leur apparition, ainsi que finaliser le level design. De plus, en tant que game designer du projet, une fois que tout sera implémenté je repasserai dans les crafts, items, mobs, stats... Afin d'essayer d'équilibrer le jeu car pour l'instant beaucoup de choses sont implémentées sans forcément considérer les autres aspects du jeu. Et enfin, biensûr si j'ai encore du temps, je le consacrerai à essayer de finir le jeu coûte que coûte si certaines parties ne sont pas finalisées, car je tiens vraiment à ce que ce projet voit le jour.

4 Antoine

4.1 Ennemis

4.1.1 Entités

Pour gérer les entités, j'ai implémenté une classe très basique : `Entity`, qui contient les informations sur l'entité comme ses statistiques. Les statistiques sont gérées grâce aux `FlatStats` et la classe `Entity` gère toute seule un cycle de régénération. En effet, chaque seconde, toutes les statistiques dont le nom finit par « -regen » ajoute leur valeur à la statistique associée. Par exemple, « hp-regen +3 » va régénérer la statistique « hp » de 3 points par seconde. Deux instances `FlatStats` sont stockées dans chaque entité : `BaseStats`, « le plafond » et `CurrentStats` qui représente les statistiques courantes de l'entité. Cette classe offre aussi la possibilité d'appliquer des effets comme ceux une potion ou plus tard des effets de sorts. Les potions sont utilisées pour deux cas de figure différents. Quand elles ont une durée de 0, les statistiques de la potion vont « recharger » `CurrentStats` sans pour autant dépasser celles de `BaseStats`. Si la potion consommée a une durée de x secondes, alors ses statistiques seront ajoutées à `BaseStats` et à `CurrentStats` pendant x secondes. Par exemple une potion avec « hp-regen +3 » et une durée de 5 secondes agira comme une potion de régénération qui incrémentera hp de 3 chaque seconde pendant 5 secondes. Pour afficher les statistiques, `Entity` implémente une fonction `GetStatsString()` qui renvoie une chaîne de caractère prête à être affichée comme dans montré dans Figure 26. Finalement, la classe `Entity` contient une méthode clef pour le mouvement : `Move()`

FIGURE 26 – Statistiques du joueur dans l'inventaire

4.1.2 Collisions

En effet, la mission de cette fonction `Move()` est de gérer les collisions. Tout d'abord, pour se déplacer dans un monde en trois dimensions comme celui d'Hermita, un simple déplacement en 2 dimensions ne suffit pas pour donner l'illusion recherchée par le point de vue isométrique. De plus il faut gérer les collisions pour pouvoir implémenter une recherche de chemin qui fasse sens et un joueur qui ne puisse pas marcher dans le vide. Nos besoins étant peu communs, il n'y a pas de solution toute faite pour gérer les collisions comme on le voulait ; Pas de tutoriel non plus. Il a donc fallu réinventer la roue et tout implémenter avec du code. Les collisions sont inspirées des collisions avec les dalles de *Minecraft*. Une dalle peut être vue comme la moitié d'un cube dans sa verticalité. (Figure 27)

FIGURE 27 – Une dalle dans notre jeu

Le joueur peut se déplacer si la différence de hauteur entre la dalle sur laquelle il se trouve et la dalle sur laquelle il veut se déplacer est inférieure à 1. De plus, si un joueur se trouve au milieu de 4 dalles, c'est la dalle la plus haute qui détermine la hauteur du joueur. Dans l'exemple de la Figure 28, le joueur se trouve sur la dalle la plus haute.

FIGURE 28 – Le joueur positionné sur quatre dalles

Similairement à *Minecraft*, si le joueur rencontre une différence de hauteur égale à 1, sa position est instantanément modifiée pour qu'il soit à la bonne hauteur. Cela permet de s'éviter une animation de saut et offre une meilleure expérience de jeu pour le joueur. Cependant, ce changement brusque de position est gênant quand la caméra est attachée au joueur puisque la caméra subit des à-coups qui peuvent être désagréables. Pour éviter cela, j'ai implémenté un petit script pour la caméra qui utilise `Vector3.SmoothDamp()` pour que la caméra se déplace progressivement vers la position du joueur à une vitesse limitée.

En conclusion, ces collisions permettent un déplacement agréable pour le joueur puisqu'il est complètement libre de se déplacer dans le monde là où beaucoup de jeux isométriques 2D limitent le joueur à un déplacement sur une grille comme dans *Dofus*. De plus comme ce système utilise la *tilemap* du monde, le Game Designer n'a pas besoin de gérer les collisions à la main comme certaines solutions suggérées par *Unity*.

4.1.3 Pathfinding

Une fois les collisions gérées, il est possible d'implémenter un algorithme de recherche de chemin. Pour cela, j'ai utilisé l'algorithme A*. Le principe est simple mais certains détails sont à prendre en compte pour que l'algorithme fonctionne correctement dans un monde isométrique. D'abord pour la distance, la diagonale verticale d'une dalle est 2 fois plus courte que l'horizontale

(Figure 27) mais dans le monde théorique, la diagonale est la même que l'horizontale. De plus, la différence de hauteur ne compte pas dans le coût du déplacement puisque les changements de niveau se font instantanément. Aussi, il a fallu changer les directions qu'essaient de prendre les entités pour que l'algorithme fonctionne correctement. En effet, dans un monde isométrique, les entités peuvent se déplacer dans 8 directions et non 4 comme dans un monde 2D. Et parcourir la même distance en diagonale et en ligne droite n'est pas une bonne idée car *gauche ; haut ;* n'arrive pas au même endroit que *haut+gauche ;* (Figure 29) et donc l'algorithme ne fonctionnerait pas correctement. Quand on parcourt $\sqrt{2}$ pour aller en diagonale, le résultat est bien plus probant comme on peut le voir sur la Figure 30.

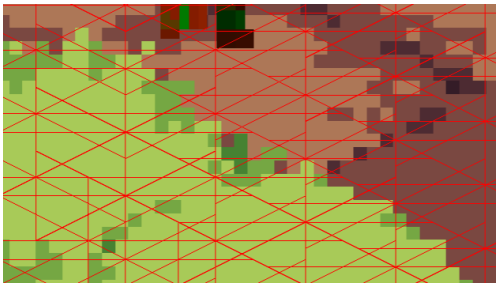


FIGURE 29 – Parcours non adapté

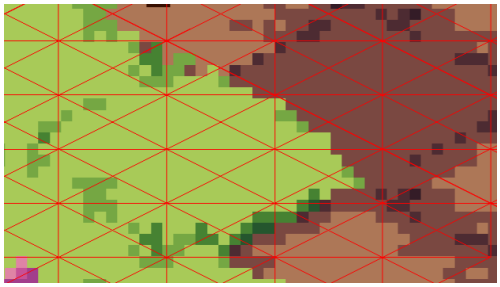


FIGURE 30 – Parcours adapté

Pour le reste de l'implémentation, il s'agit d'un classique algorithme A* avec une liste de priorités pour les nœuds à explorer. Pour ce faire, j'ai créé la classe `Node` qui implémente l'interface `IComparable` pour pouvoir être utilisée dans la liste de priorités. Une fois le chemin reconstruit (Figure 31), la suite des étapes est renvoyée sous forme de liste de `int` qui correspondent aux directions à prendre. Sur l'image on peut voir que le *gobelin* évite bel et bien le mur qui se trouve entre lui et le joueur. Il est aussi possible de voir que le monstre n'atteint pas parfaitement la position du joueur. Cela dépend de la distance recherchée entre le monstre et le joueur. C'est un paramètre que le Game Designer peut modifier pour que le monstre soit plus ou moins agressif. Si d'aventure un monstre attaquait à distance comme un archer, il suffit de modifier ce paramètre pour que le monstre soit plus ou moins loin du joueur.

4.1.4 Optimisation

Dans la recherche de chemin, une optimisation nécessaire consiste à faire avancer le monstre à grands pas. Au lieu de faire des petits déplacement, le chemin exploré est parsemé est très espacé comme on peut le voir dans la Figure 31. Cependant, cela réduit la précision du déplacement du monstre. C'est un simple paramètre qui peut être changé dans le temps en fonction des besoins mais actuellement, aucun bug n'a été rencontré avec cette valeur.

Toujours pour la recherche du chemin, ma première vraie idée pour optimiser le calcul du chemin était d'exécuter la recherche dans un thread séparé. Cependant, j'ai rencontré cette erreur :

```
UnityException: get_transform can only be called from the main thread.
```

Contraint à rester dans le thread principal, j'ai tout de même gardé une fonction qui renvoie un objet `Task` du C#. L'utilité n'est que de rendre la fonction asynchrone et de pouvoir continuer l'exécution du thread principal en parallèle. Cela permet de ne pas bloquer le thread

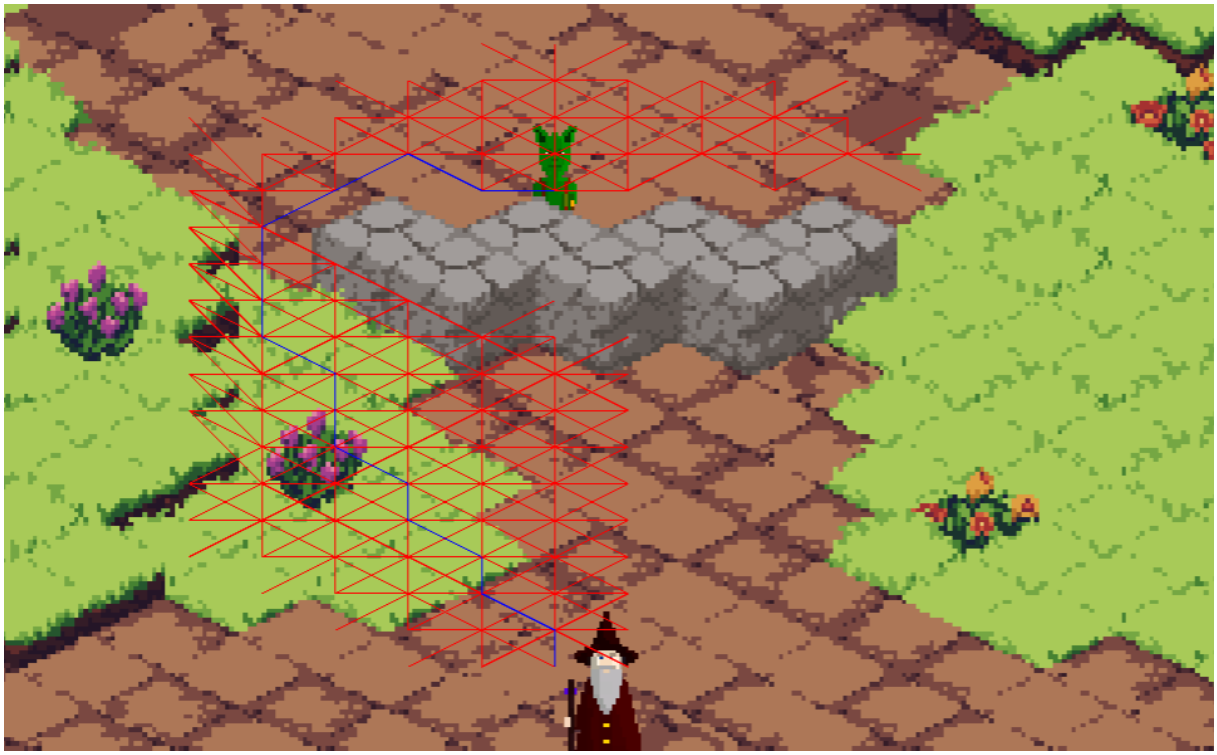


FIGURE 31 – Chemin reconstruit

principal pendant le calcul du chemin grâce à `await Task.Yield();`. Cette implémentation est particulièrement utile quand le nombre de monstres augmente.

Finalement, j'ai aussi implémenté une sorte de « désynchronisation » des monstres. En effet, si tous les monstres commencent à chercher un chemin en même temps, cela peut ralentir le jeu. Pour éviter cela, j'ai ajouté un délai aléatoire avant de commencer la recherche de chemin. Cela permet de ne pas avoir tous les monstres qui cherchent un chemin en même temps et donc de ne pas ralentir le jeu.

4.1.5 Barre de vie

La barre de vie est simplement un *Slider* d'*Unity* qui est positionné au-dessus de la tête du monstre. Pour cela, il suffit de récupérer la position du monstre, de la modifier en fonction de sa taille et de la positionner au-dessus de sa tête. Cependant, il faut faire une lien entre la position dans le monde et la position dans l'interface. Pour cela, j'ai utilisé la fonction `camera.WorldToScreenPoint` qui permet de convertir une position dans le monde en position dans l'interface. Les joueurs partagent le même système de barre de vie. Il suffit de changer la couleur et de faire en sorte que la barre ne soit affichée que si ce n'est pas le personnage du joueur. Des teintes rouges sont associées aux monstres et des teintes vertes pour les alliés. Vous pouvez voir un exemple dans la Figure 32 avec un nom factice en attendant une implémentation du système de sauvegarde.

4.1.6 Attaque

C'est bien si le monstre peut nous trouver mais il faut aussi qu'il puisse nous attaquer. Pour cela, dans la boucle où le monstre se déplace, si le monstre est arrêté et qu'il est assez

FIGURE 32 – Barres de vie

proche du joueur, il lance une fonction d'attaque en coroutine. `StartCoroutine(Attack());` Cette fonction d'attaque est très simple. Après avoir infligé des dégâts au joueur en fonction de sa statistique d'attaque, le monstre attend un temps défini par le Game Designer et répète l'opération jusqu'à ce que le monstre soit trop loin du joueur. La direction de son animation d'attaque est cependant définie en fonction de la dernière direction de déplacement du monstre. Ce détail changera peut-être d'ici à la dernière soutenance. Il reste encore un autre détail au niveau de l'animation, tant que le monstre attaque, son animation (voir Figure 33) tourne en boucle alors qu'elle pourrait être synchronisée avec les dégâts infligés au joueur.

FIGURE 33 – Monstre qui attaque

4.1.7 Synchronisation multijoueur

Le multijoueur a été une source de long debug pénible pour faire en sorte que les monstres soient synchronisés, comme on peut le voir dans la Figure 34, chez tous les joueurs sans avoir à calculer le chemin sur chaque ordinateur par exemple. J'ai décidé de tout mettre sur l'or-

FIGURE 34 – Synchronisation des monstres

dinateur du joueur qui héberge. C'est le « host » qui calcule la vie des entités, c'est lui qui calcule les chemins et fait bouger les monstres. Seul inventaire et mouvement des joueurs sont gérés côté client. Cette décision demande de faire attention à ne pas trop surcharger l'ordinateur du host qui pourrait ralentir le jeu pour les autres joueurs. Cependant, cela simplifie grandement la synchronisation des monstres et allège le traffic réseau. Par exemple, plutôt que de partager chaque statistique de chaque monstre, le serveur envoie simplement la vie et le reste des statistiques n'est pas nécessaire pour le client. Par exemple la vie est partagée grâce à une `NetworkVariable<float>` et le serveur envoie les mises à jour de cette manière : `OnStatsChange += () => health.Value = CurrentStats["hp"];` pour envoyer la vie du monstre dès qu'il y a un changement.

4.1.8 Apparition

L'apparition des monstres est gérée par des « tanières », la classe `Lair`. Qui est encore assez simple pour le moment. Si un joueur entre dans un rayon donné, un nombre aléatoire est généré entre les bornes définies par le Game Designer et ce nombre de monstres apparaissent. Si un joueur sort du rayon, les monstres disparaissent. C'est une fonctionnalité qui peut-être améliorée dans le futur avec un système de sauvegarde des monstres qui ont été tués par exemple. Cependant, pour ce qui est des *inputs*, j'aimerais garder la même simplicité pour le Game Designer qui a une interface comme dans la Figure 35 pour définir les tanières. Ces pistes seront explorées quand le système de sauvegarde sera implémenté. Un petit détail ici consiste à vérifier que ce soit bien le joueur qui héberge qui fasse apparaître les monstres. Mais les Lairs ne sont pas des NetworkBehaviour et donc ne peuvent pas accéder à la variable `isHost`. Pour contourner ce problème, on doit utiliser la propriété `NetworkManager.Singleton.IsHost`. Et pour finir, une fois le monstre instancié du côté hébergeur, il suffit d'appeler la fonction `Spawn` de la classe `NetworkObject` pour que le monstre soit instancié chez tous les clients.

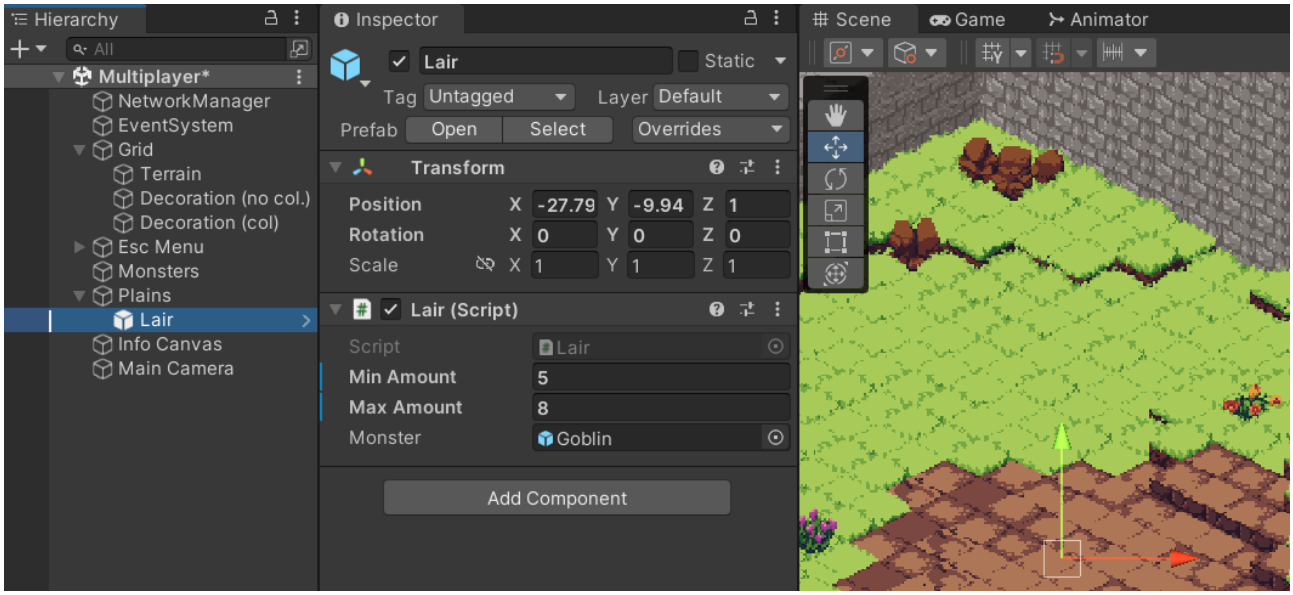


FIGURE 35 – Création d'une tanière dans *Unity*

4.2 Inventaire

4.2.1 Favoris et menu contextuel

Pour aider le joueur à gérer son inventaire, j'ai implémenté un système de favoris. En faisant un clic droit sur un objet, le joueur peut l'ajouter aux favoris. C'est d'ailleurs aussi avec le clic droit que le joueur peut se débarrasser d'objets. Pour afficher les options du clic droit, il suffit d'un panneau qui s'affiche à la position de la souris comme on peut le voir dans la Figure 36. Trois options sont disponibles : *Ajouter aux favoris*, *Détruire un objet* et *Détruire toute la pile*. Afin de simplifier l'expérience utilisateur, un clic en dehors du panneau ferme le panneau. De plus, si le joueur détruit un objet et qu'il n'y a plus d'objet dans la pile, le panneau se ferme automatiquement. Si l'objet est un équipé par le joueur, il ne peut pas être détruit.

Les favoris ont pour le moment comme seule fonction d'être affichés en premier dans l'inventaire mais le tri des objets peut toujours être personnalisé, cela agira simplement comme si c'était 2 listes différentes d'objets. (voir Figure 36)

4.2.2 Rareté

Afin de motiver le joueur à chercher des objets rares, j'ai ajouté une touche graphique qui indique la rareté de l'objet. Il s'agit simplement d'une image en plus de l'icône de l'objet et la couleur de l'objet change en fonction de la rareté. Une couleur est associée à chacun des tiers de rareté : *S, A, B, C et D*. Les couleurs reprennent les codes classiques du jeu vidéo mais sont facilement modifiables si ce n'est pas au goût du Game Designer. Un exemple de ce système est visible dans la Figure 37.

FIGURE 36 – Menu contextuel et favoris

FIGURE 37 – Équipements triés par tier de rareté

4.3 Multijoueur

4.3.1 Écran de chargement

Parfois, les petits détails font beaucoup. Afin de rendre l'attente plus agréable, j'ai ajouté un écran de chargement qui s'affiche pendant que les joueurs se connectent au serveur. Il s'agit d'un ciel en pixel art dont les nuages se déplacent avec un effet de parallaxe. Au premier plan, on peut voir le personnage du joueur qui joue l'animation de course. (voir Figure 38) Cette animation est aussi utile pour s'assurer que le joueur ne clique par sur des boutons pendant le chargement. Un point faible de cette solution est qu'elle commence à s'arrêter au bout de 20 secondes. Mais en pratique, le temps de chargement dure rarement plus de 5 secondes donc ce n'est pas un problème. Bien que la réalisation de cet écran de chargement ait été assez simple, c'est avec elle que j'ai appris à bien utiliser les animations dans *Unity*.

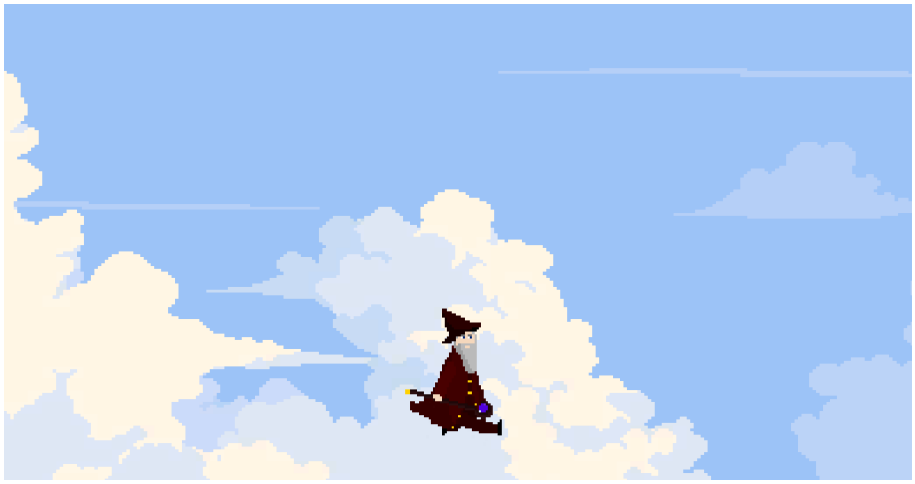


FIGURE 38 – Écran de chargement

4.3.2 Gestion d'erreurs

Dans un monde parfait, on se concentre sur ce qui marche et on le fait marcher encore mieux. Mais dans la vraie vie, Alice va se tromper de mot de passe, Bob va essayer de cliquer sans code et Charlie n'aura pas de connexion internet. Pour tous ces olibrius, il faut prévoir des messages d'erreur qui expliquent ce qui ne va pas. Vous pouvez voir dans la Figure 39 un exemple de message d'erreur qui s'affiche si le joueur n'a pas entré de mot de passe.

FIGURE 39 – Message d'erreur

Pour vérifier si le joueur est connecté à internet, j'utilise `UnityWebRequest` en faisant une requête à *http ://google.com* et en vérifiant si ça ne génère pas d'erreur. Le reste consiste simplement à de petits tests insérés dans le code. Ces petits détails permettent de fournir un produit qui paraît plus fini et en cas de problème, on aura quelque chose à montrer pendant la soutenance même si nous n'avons pas de réseau en amphithéâtre.

4.3.3 Interface en jeu

Jouer avec ses amis est l'essence même de notre jeu. Pour rendre cette expérience fluide, il faut que toutes les opérations liées au multijoueurs se déroulent de manière très fluide. Par exemple, taper le mot de passe à chaque fois que l'on veut rejoindre une partie ou quand on veut envoyer un code est très pénible. J'ai rajouté une interface qui s'ouvre avec la touche *Échap* que

l'on peut voir dans la Figure 40. Pour éviter cela, j'ai transformé le code qui s'affichait en haut à droite en un bouton qui copie le code dans le presse-papier. Ceci est facilement réalisable avec `GUIUtility.systemCopyBuffer = "ABCDEF";` et permet de gagner du temps. On peut aussi trouver dans cette interface des boutons pour quitter la partie ou quitter le jeu.

FIGURE 40 – Interface en jeu

4.4 Site web

4.4.1 Page d'accueil

Mon objectif ici était de faire une page très épurée avec des éléments originaux. Il s'agit d'une page classique avec quatre sections dont trois qui présentent les différents aspects du jeu. La première section donne juste le nom du jeu et le nom du groupe. Comme on peut le voir dans la Figure 41, la section est composée d'un simple texte avec des dalles (voir 27) qui se déplacent en arrière-plan avec le scroll et avec des vitesses différentes.

Chaque section est associée à des animations de dalles différentes. La section sur le combat fait sauter les dalles avec une animation brusque pour inspirer l'action. Cet effet est réalisé grâce à une courbe de Bézier générée avec *https ://cubic-bezier.com*. La section sur l'aspect d'aventure fait bouger les dalles dans différentes directions de façon douce pour inspirer la découverte. La section sur le crafting fait changer la taille des dalles pour inspirer un sentiment de construction. En plus de cela, les animation possèdent des petites animations d'apparition et de disparition quand l'utilisateur scroll.

4.4.2 Page de téléchargement

Pour les gens qui veulent aller vite, il faut quelque chose de concis où tout est rapide et intuitif. Comme vous pouvez le voir dans la Figure 42, la page de téléchargement est très simple. Une tentative a été faite avec des images en pixel art mais si les parchemins offrent un rendu satisfaisant, les icônes des systèmes d'exploitation ne sont pas aussi réussies. Ceci est amené

FIGURE 41 – Page d'accueil

à changer dans le futur. L'image des parchemins est générée par *DALL-E* et retouchée dans *Photoshop*.

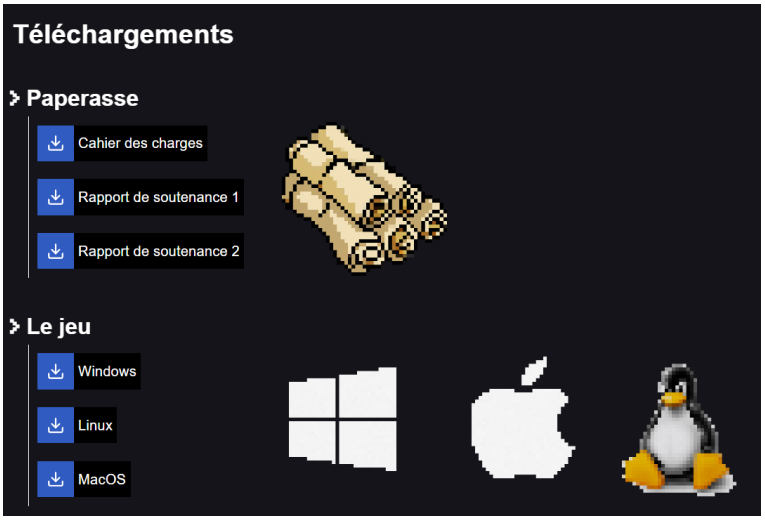


FIGURE 42 – Page de téléchargement

4.4.3 Performaces

Le site web est hébergé gratuitement, des effets très gourmands en ressources ont été implémentés pour une visuel marquant et agréable. Malgré tout cela, le site se débrouille très bien avec un score plus que satisfaisant sur *Google Lighthouse* comme on peut le voir dans la Figure 43. Ceci est rendu possible grâce à l'utilisation d'*Astro* qui réduit très fortement le temps de chargement des pages avec des méthodes de *Server Side Rendering* et de *Static-Site Generating*. On rencontre rarement des sites aussi rapides avec des simulations de fluides et des

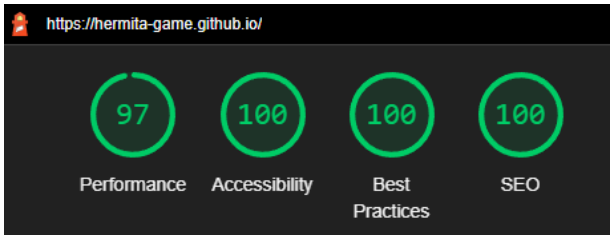


FIGURE 43 – Performances du site

animations complexes associées au scroll. Ces résultats justifient donc au final l'idée de ne pas utiliser *Sveltekit* malgré sa fonctionnalité intéressante de *Single Page Application*.

4.5 Bonus

4.5.1 Musiques

Un jeu sans musique c'est comme un steak sans sel... Par chance, nous avons un ami compositeur dans son temps libre qui s'est proposé pour faire les musiques de notre jeu. Il a donc réalisé nombre de musiques pour différentes zones du jeu. Nous pouvons nous estimer vraiment chanceux parce que toutes les musiques ont été pensées spécifiquement dans notre jeu et s'inscrive parfaitement dans notre univers. Un grand merci à Henri Buissière qui nous a offert son travail. Je me suis donc occupé d'implémenter les pistes du *Housing* et de la première zone. Le principe est simple, jouer en boucle les musiques à la suite avec des pauses de 2 secondes entre chaque. À l'avenir, on pourrait implémenter une fonctionnalité qui permet de définir des zones simplement pour changer les musiques.

4.6 Prochaine Soutenance

Pour la prochaine soutenance, je prévois de réaliser la page des crédits où l'on retrouvera les membres de l'équipe et nos aides extérieures. Je prévois aussi d'améliorer le fonctionnement des tanières et de régler d'éventuels bugs aves les IA. De plus, la possibilité d'héberger et de rejoindre une partie est parfaitement fonctionnelle. Finalement, pour ce qui est de l'inventaire, il reste la possibilité d'améliorer l'apparence mais il est fonctionnel. Ayant peu de tâches lourdes, une bonne partie de mon travail consistera à aider mes collègues à régler les soucis puisque beaucoup d'éléments dépendent des briques élémentaires que j'ai implémentées.

5 Anouar

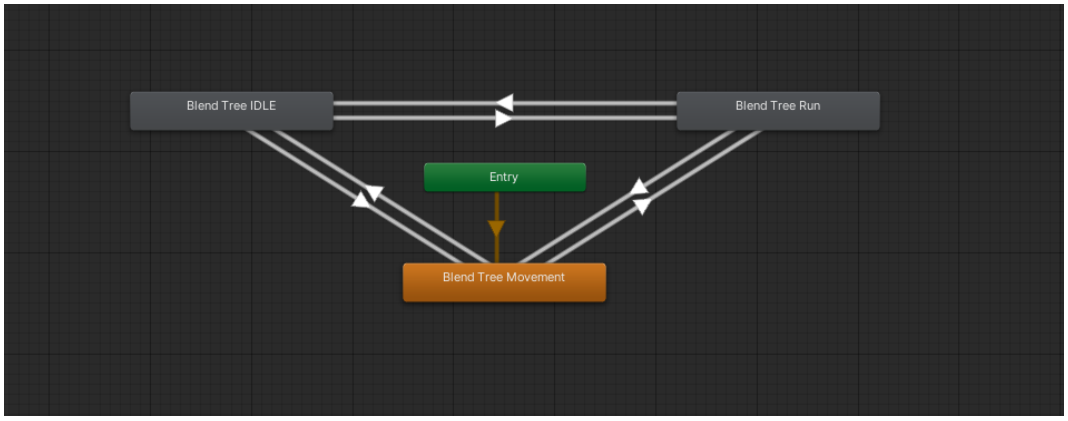
Pour cette deuxième soutenance j'ai dû m'occuper de finir les déplacements, de réaliser le système de sort et d'implémenter une interface en jeu incluant les emplacements pour les sorts, une barre de vie et une barre de mana. C'est sûrement la soutenance ou j'ai rencontré le plus de difficultés et ce de très loin.

5.0.1 Difficultés

Dès la fin de la première soutenance, j'ai ressenti une très forte baisse de motivation en ce qui concerne ce projet. En effet, j'ai de plus en plus l'impression que celui-ci ne m'appartient

plus. Je me réalise ainsi comme un exécutant qui a de moins en moins son mot à dire. Je regrette profondément le choix de réaliser un jeu en vue isométrique car en plus de poser énormément de problème et ce à toute l'équipe, je trouve que cela ne permet pas d'avoir un rendu aussi propre que ce à quoi j'aspirais. Lorsque j'ai proposé un changement sur ce point, le groupe a rejeté ma proposition en invoquant le fait que trop de travail avait été fourni dans ce sens. J'ai donc entendu cet argument mais je pense que cet élément est à l'origine d'une sorte de rupture quant à ma motivation. Ce qui a été déterminant dans mon implication limitée quant aux tâches que j'avais à réaliser pour cette soutenance fut le retour dans ma vie de certain problèmes personnels qui, je le pensais, appartenaient au passé. Durant la période d'entre-deux soutenance, je n'ai pas pu être aussi communicatif que je l'aurais voulu, j'ai commencé progressivement à m'isoler tant de mon groupe que de ma vie et je me suis retrouvé dans une phase de décrochage exacerbée par ce que j'ai aussi vécu comme un long silence en ce qui concerne le projet de toute part dans le sens où j'avais l'impression que nous ne parlions presque plus du projet. Je sais qu'il était de ma responsabilité de mener ce projet à bien et que le groupe de mérite pas de subir les problèmes que je peux traverser mais je craignais de m'exprimer dans un climat où je ne me sentais plus vraiment à ma place.

5.1 Gestion des déplacements



Par rapport à la dernière soutenance, j'ai complété le système de déplacement de mon personnage principal en y ajoutant un dash et une course. Cette fonctionnalité donne aux joueurs une plus grande liberté de mouvement et leur permet de se déplacer plus rapidement sur la carte. Le dash permet au joueur de réaliser une petite accélération dans une direction donnée. Cependant, après chaque utilisation, il y a un temps de recharge qui empêche le joueur de l'utiliser à nouveau immédiatement. Cela permet de maintenir un certain équilibre dans le jeu et empêche les joueurs d'utiliser cette fonctionnalité de manière excessive. En ce qui concerne la course, elle permet au joueur de se déplacer plus rapidement que la vitesse de déplacement normale. L'ajout de ces fonctionnalités a été un défi, mais j'ai pu les intégrer de manière transparente au système de déplacement existant. Je suis satisfait du résultat et je pense que les joueurs apprécieront la plus grande liberté de mouvement offerte par ces fonctionnalités. Pour ajouter le dash avec cooldown, j'ai dû ajouter une variable booléenne qui indique si le dash est en cours d'utilisation ou non, j'ai mis en place une coroutine qui enregistre le moment où le dash a été utilisé. J'ai également dû mettre en place une fonction de recharge qui utilise une autre variable de temps pour s'assurer que le joueur ne peut pas utiliser le dash

de manière répétitive. Quant à la course, j'ai ajouté une autre variable booléenne pour suivre si le personnage est en mode de course ou non. Si la course est activée, j'augmente simplement la vitesse de déplacement du personnage, l'idée est que le personnage se déplace en marchant dans le housing mais en courant en multijoueur. J'ai également ajouté l'animation pour la course. Cela aide le joueur à comprendre ce qui se passe et ajoute une touche de réalisme à l'expérience de jeu. Pour assurer une jouabilité fluide, j'ai testé ces fonctionnalités en les intégrant à des scènes plus grandes et plus complexes, en ajustant les variables et les temps de recharge en conséquence pour qu'ils soient adaptés au jeu dans son ensemble.

5.2 Système de sort

Tout d'abord, j'ai créé un inventaire de sorts en bas de l'écran qui affiche les quatre sorts disponibles. J'ai mis en place un système de sélection de sort qui permet au joueur de cliquer sur un sort pour le sélectionner. Ensuite, j'ai ajouté une barre de vie et une barre de mana en haut à gauche de l'écran pour aider le joueur à suivre la santé et la capacité magique de son personnage. La barre de vie se vide lorsque le personnage est attaqué, tandis que la barre de mana diminue lorsqu'un sort est lancé. Pour lancer un sort, le joueur doit d'abord sélectionner le sort qu'il souhaite utiliser. Une fois le sort sélectionné, le joueur peut cliquer dans la direction qu'il souhaite pour lancer le sort. J'ai utilisé des scripts de raycasting pour déterminer la direction dans laquelle le joueur a cliqué et pour lancer le sort dans cette direction. Pour rendre le système de lancer de sort plus interactif, j'ai ajouté des effets visuels pour chaque sort. Par exemple, lorsqu'un joueur utilise un sort de feu, une boule de feu apparaît à l'endroit où le sort est lancé. De plus, j'ai ajouté des effets sonores pour chaque sort afin d'améliorer l'immersion du joueur. Pour m'assurer que le système de lancer de sort est équilibré, j'ai testé chaque sort en ajustant leur puissance et leur consommation de mana. De plus, j'ai implémenté un temps de recharge pour chaque sort afin que le joueur ne puisse pas utiliser le même sort de manière répétitive. Enfin, j'ai intégré ce nouveau système de lancer de sort à mon système

de combat existant pour offrir aux joueurs une expérience de jeu plus riche et plus variée. Tout d'abord, j'ai créé un inventaire de sorts en utilisant un canvas dans Unity. J'ai utilisé des boutons pour représenter chaque sort disponible, avec une image pour chaque sort et un petit texte pour le coût en mana. Pour détecter la direction dans laquelle le joueur a cliqué, j'ai utilisé une méthode appelée "Raycasting". Cette méthode crée un rayon imaginaire à partir du point de départ (dans ce cas, le personnage du joueur) qui suit la direction du clic de souris jusqu'à ce qu'il frappe un objet dans le monde du jeu. En utilisant les coordonnées de l'objet frappé, j'ai pu déterminer la direction dans laquelle le joueur a cliqué et lancer le sort dans cette direction. J'ai implémenté des calculs trigonométriques afin de pouvoir gérer la rotation des sprites de sorts en fonction de l'angle de tir et de la position du joueur. Pour suivre la santé et la capacité magique du joueur, j'ai utilisé des barres de progression. J'ai créé deux barres de progression pour chaque joueur : une pour la santé et une pour la capacité magique. J'ai également ajouté un script de gestion de la santé et de la capacité magique qui met à jour les barres de progression en temps réel en fonction de l'état du joueur. Dans l'ensemble, la création de ce système de lancer de sort a nécessité une planification minutieuse, une programmation détaillée pour assurer une expérience de jeu cohérente et immersive pour le joueur.

En conclusion, l'ajout du système de déplacement et du système de lancer de sort à mon jeu en a été une expérience enrichissante sur de nombreux niveaux. J'ai pu mettre en pratique mes compétences en programmation de jeu pour créer une expérience de jeu fluide et immersive pour les joueurs. Le système de déplacement a été la base de mon jeu, permettant aux joueurs de naviguer dans l'environnement et de découvrir le monde qui les entoure. L'ajout du système de lancer de sort a permis aux joueurs de prendre part à des combats passionnants et de relever des défis plus complexes. Ces deux systèmes se sont complétés pour créer une expérience de jeu complète et engageante. En travaillant sur ces systèmes, j'ai également pu développer des compétences en résolution de problèmes, en planification de projet et en collaboration. En fin de compte, l'ajout de ces systèmes à mon jeu a permis de créer une expérience de jeu plus riche et plus immersive, offrant aux joueurs des heures de divertissement et de plaisir. Pour la prochaine soutenance je ferais en sorte de compléter au maximum le fonctionnement des sorts et je m'attèlerai sérieusement à la gestion de la sauvegarde. Surtout, je m'engage à travailler en synergie avec mon groupe afin que mes contributions ne soient plus invisibilisées.

6 Conclusion

Le projet Hermita poursuit sa lancée, ainsi nous sommes fiers de dire que nous avons travaillé dur pour atteindre nos objectifs. Même si certain problèmes de communication ont pu se faire sentir. Nous sommes conscients que chaque membre de notre équipe a des compétences et des idées uniques qui ont contribué à notre succès jusqu'à présent. Nous sommes déterminés à continuer à travailler ensemble de manière efficace et transparente, en communiquant clairement et en étant ouverts aux commentaires de chacun. En gardant à l'esprit que l'importance de la communication dans le travail d'équipe ne doit jamais être sous-estimée. C'est seulement grâce à une communication ouverte et honnête que nous pourrons continuer à avancer ensemble vers nos objectifs communs.