Rapport de projet EPITA - *Projet S2*

HERMITA *Produit par Epistars™*

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

Table des matières

1 Introduction

Cette section aura la charge de vous présenter les origines de notre société mais aussi et surtout l'équipe qui s'attelera à mener à bien le projet de jeu que nous souhaitons réaliser.

1.1 Mise en contexte

1.1.1 Epistars™ : Les origines

La société Epistars a vu le jour lorsqu'un groupe d'amis, d'un naturel introverti, se sont rapprochés et ont créé de solides liens. Dès leur rentrée dans une école d'informatique réputée, ce groupe de quatre membres s'est vite formé durant leur journée d'intégration dans le but de devenir collègues pour réaliser un travail ambitieux sur un jeu vidéo. Le nom de la société fut choisi au cours de cette même journée pour une chasse au trésor en équipe, en fan de Star Wars épitéens ce nom était tout trouvé : Epistars (la fig. 1 illustre notre logo).

Cette proximité nous permet d'avoir une équipe soudée qui sait avancer sans contradictions. Dotée d'une forte motivation, nous avons la grande ambition de faire de la société Epistars™ un acteur majeur du monde vidéoludique dans les années à venir. Notre équipe est constituée de membres pourvus d'un riche ensemble de fortes individualités qui, de par leur complémentarité, forment un groupe solide et techniquement peformant.

1.1.2 Présentation de l'équipe

Notre équipe est composée des membres suivants :

- **BAPTISTE DURRINGER** (***Lead Game Designer***) : hautement compétent, fort de sa longue expérience dans le milieu vidéoludique.
- **DYLAN DE ARAUJO** (**Directeur Artistique**) : plein de talent en raison de sa passion dévorante pour toutes les formes d'art depuis son plus jeune âge.
- **ANTOINE BLUMENROEDER** (**Directeur d'Architecture**) : féru de maths depuis toujours et doté d'une exceptionnelle intuition dans la modélisation de problèmes.
- **ANOUAR BELMEDJENNEH** (**Directeur Technique**) : fort de ses expériences acquises et de sa maîtrise des différents outils, est le plus qualifié pour diriger ce projet.

Nous aurons à cœur tout au long du processus de développement d'offrir une prestation de qualité.

1.1.3 Contexte du projet

La société Epistars™ a choisi de mettre ses compétences au service de l'École Pour l'Informatique et les Techniques Avancées dans le but de mener à bien la réalisation d'un logiciel de loisir type jeu vidéo pour la validation de ce qui est appelé en interne « Projet de S2 ». Cette opportunité nous permettra d'affiner nos compétences et notre technicité, mais aussi et surtout de renforcer notre capacité à travailler en équipe sur un projet complet.

1.1.4 Objet de l'étude

L'idée de la réalisation de ce jeu nous est venue rapidement dès que nous avons pris conscience de la liberté dont on dispose. N'ayant pas pour habitude de réaliser un projet d'une

telle envergure, nous avons choisi d'être plus ambitieux car nous croyons en son potentiel. Nous faisons le choix de faire un jeu en deux dimensions plutôt qu'en trois, ce qui nous permettra de nous focaliser sur la création d'un contenu riche, au sein d'un univers original.

Nous voulons réaliser un jeu qui oppose action nerveuse et quiétude reposante car cette alchimie nous a séduit. Étant tous dotés d'un goût pour l'aventure et le défi, ce projet a trouvé écho dans nos cœurs. De plus, nous avons tous une certaine nostalgie pour les jeux de notre enfance. Ce sera l'occasion de retranscrire ce qui nous faisait vibrer plus jeunes tout en y ajoutant un regard plus mature. L'objectif est d'utiliser pleinement les différentes possibilités offertes par le format du jeu vidéo.

1.1.5 Nature du projet

Ce projet consiste en un jeu que nous réaliserons dans un style pixel art, il se présente sous la forme d'un monde de *fantasy* fait d'îlots flottants (avec de légères touches *steampunk*) en deux dimensions avec une vue isométrique. Notre jeu aura une composante *housing* et appartiendra au genre du RPG. Lorsqu'on lance le jeu, le joueur apparaît dans une zone qui lui servira de base et lui donnera accès aux autres îlots qui constituent son monde. Son objectif sera d'explorer son univers afin de réunir les ressources qui lui seront nécessaires pour faire évoluer sa base tout en ayant à combattre les créatures hostiles qui se dresseront sur sa route.

Hermita vise principalement à convaincre un public mature amateurs de jeux vidéo indépendants, avide d'aventure et d'exploration sans prise de tête. Notre jeu a pour objectif d'offrir une expérience apaisante et satisfaisante, sans que le joueur y passe tout son temps, nous voulons qu'il choisisse de le lancer pour passer un bon moment.

1.1.6 État de l'art

Le jeu que nous réalisons appartient au genre du RPG (*Role Playing Game*), très répandu dans le milieu du jeu vidéo et qui se définit par quatre principes fondamentaux : l'évolution du personnage, l'exploration, l'influence sur son monde et finalement les combats.

Le genre du RPG existe depuis maintenant plus de cinquante ans et ce bien avant l'arrivée du jeu vidéo. En effet, dès 1970, les jeux de rôles papier comme Donjon et Dragons ont inventé le concept. Ce n'est qu'en 1986, que sort le premier jeu vidéo du genre RPG : Dragon Quest sur Famicon (une console Nintendo) qui a eu un énorme succès à l'époque.

Hermita, n'a évidemment pas pour ambition de détrôner les titres établis que sont par exemple Zelda, Final Fantasy, The Elder Scrolls ou encore Dark Souls. Nous souhaitons vous présenter un jeu plus simple dans ses ambitions, offrant certes moins de possibilités qu'un triple A mais qui seront plus en adéquation avec nos objectifs et les contraintes qui nous incombent.

Pour imaginer notre jeu, nous tirons nos inspirations de différents univers pour pouvoir poser l'histoire, les décors, ou encore le gameplay. Ainsi des jeux tels que Terraria, Palia, Don't Starve (Together), Diablo ou encore League of Legends nous ont aidé à fixer le *game design* et le *level design*. Au delà des jeux vidéo, notre projet est aussi librement inspiré d'oeuvres telles que les mangas Radiant et Sword Art Online, le film d'animation Le Château dans le Ciel ou même de diverses citations de Jacques Nteka Bokolo afin de fixer l'univers, l'histoire et les décors de notre jeu.

1.2 Progression des tâches

Pour la première soutenance, nous avions bien avancé dans la globalité et nous avions même un peu d'avance sur certaines tâches sauf celles d'Anouar qui avaient déjà un peu de retard mais sans conséquence. Nous n'avions pas vraiment pris d'initiative à ce moment là et avons subit la douche froide illustrée dans la Figure 1. Nous avons eu un rendez-vous avec le coach

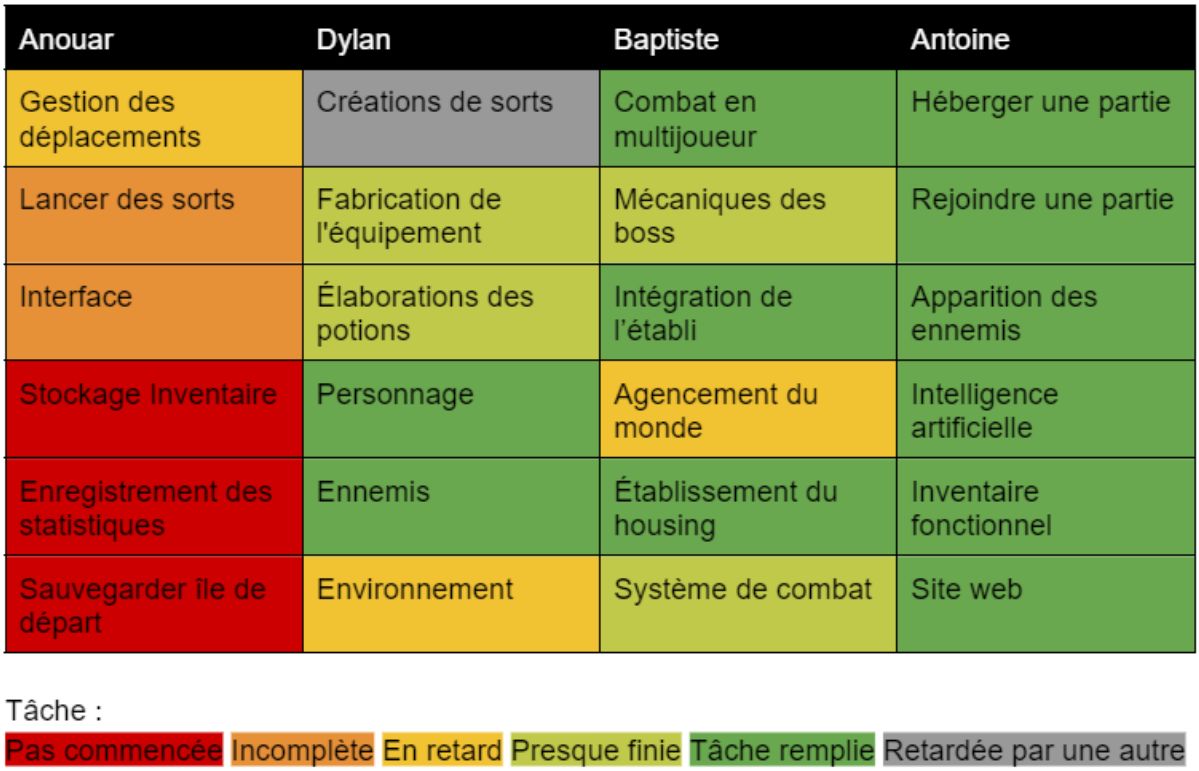

FIGURE 1 – Suivi des tâches pour la Soutenance 2

peu avant la soutenance 2 et même si les conseils n'ont pas pu être appliqué directement, 3 jours après la soutenance nous avons commencé à le faire. La première étape consistait à prendre le leadership sans être chef de projet. Ainsi, nous avons pu avancer, prendre des décisions et replanifier les tâches restantes pour mener le projet à bien. La replanification illustrée dans la Figure 2. En fin de compte, nous avons abouti à tout ce qui était prévu, sauf le menu d'accueil dont l'abandon de tâche a été communiqué 2 jours avant la dernière soutenance. Grâce aux conseils du coach, nous avons pris l'initiative d'implémenter ces tâches au fur et à mesure que leur abandon nous était communiqué.

2 Ennemis

2.1 Entités

2.1.1 Classe mère : Entity
Antoine, Soutenance 2

Anouar	Dylan	Baptiste	Antoine
Map/minimap	Finalisation des crafts	Level design	Loots des mobs
Options	Dessins entités	Implémentation des boss	Système de niveaux
Sauvegarde	Dessins environnement	Finaliser le game design	Monstres à distance
Sorts (combat et interface)	Interface	Options	Map/minimap
Menu d'accueil		Sorts (combat et interface)	Sauvegarde

Tâche :
Non réalisée par le membre | Tâche remplie | Tâche remplie et récupérée

FIGURE 2 – Replanification des tâches après la Soutenance 2

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 3. Finalement, la classe `Entity` contient une méthode clef pour le mouvement : `Move()`

2.1.2 Collisions
Antoine, Soutenance 2

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

FIGURE 3 – Statistiques du joueur dans l'inventaire

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 4)

FIGURE 4 – 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 5, le joueur se trouve sur la dalle la plus haute.

FIGURE 5 – 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*.

2.1.3 Animations personnalisées
Antoine, Soutenance 3

Pour les entités, nous avons faisions leur animations via l'animateur de Unity. Cependant, nous travaillons avec 8 directions, parfois 4, parfois 2. De plus, chaque monstre possède au moins 3 animations. Pour ce qui est du joueur, il possède 2 animations d'attaques. Cela allait devenir vite pénible pour le Game Designer de devoir créer toutes ces animations à la main. C'est pourquoi j'ai implémenté une classe `CustomAnimator` qui permet de créer des animations à partir d'une *SpriteSheet* et du nombre de directions voulues. Toutes les entités se déplacent en 8 directions, l'animateur gère les animations de manière autonome si l'entité n'a que 4 ou 2 directions pour son animation. Pour les monstres par exemple, il suffit de remplir le nom des spritesheets pour l'*idle*, la marche et l'attaque et l'animateur s'occupe du reste. Cet animateur personnalisé permet aussi de jouer une animation une seule fois avant de revenir à l'animation précédente. C'est ce qui est utilisé pour les attaques.

2.2 Monstres

2.2.1 Pathfinding
Antoine, Soutenance 2

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 4) 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 6) 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 7.

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

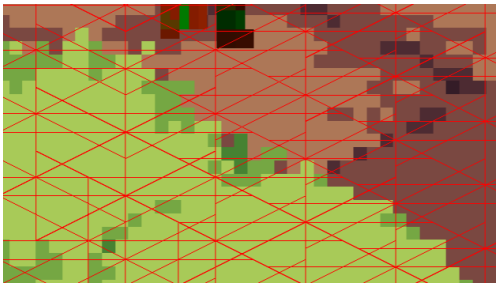

FIGURE 6 – Parcours non adapté

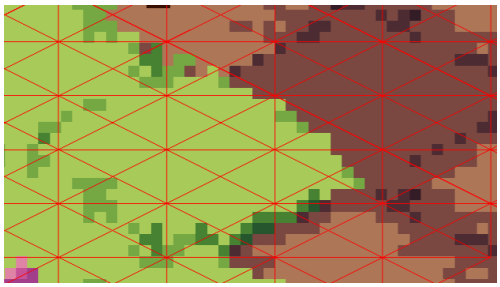

FIGURE 7 – Parcours adapté

reconstruit (Figure 8), 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.

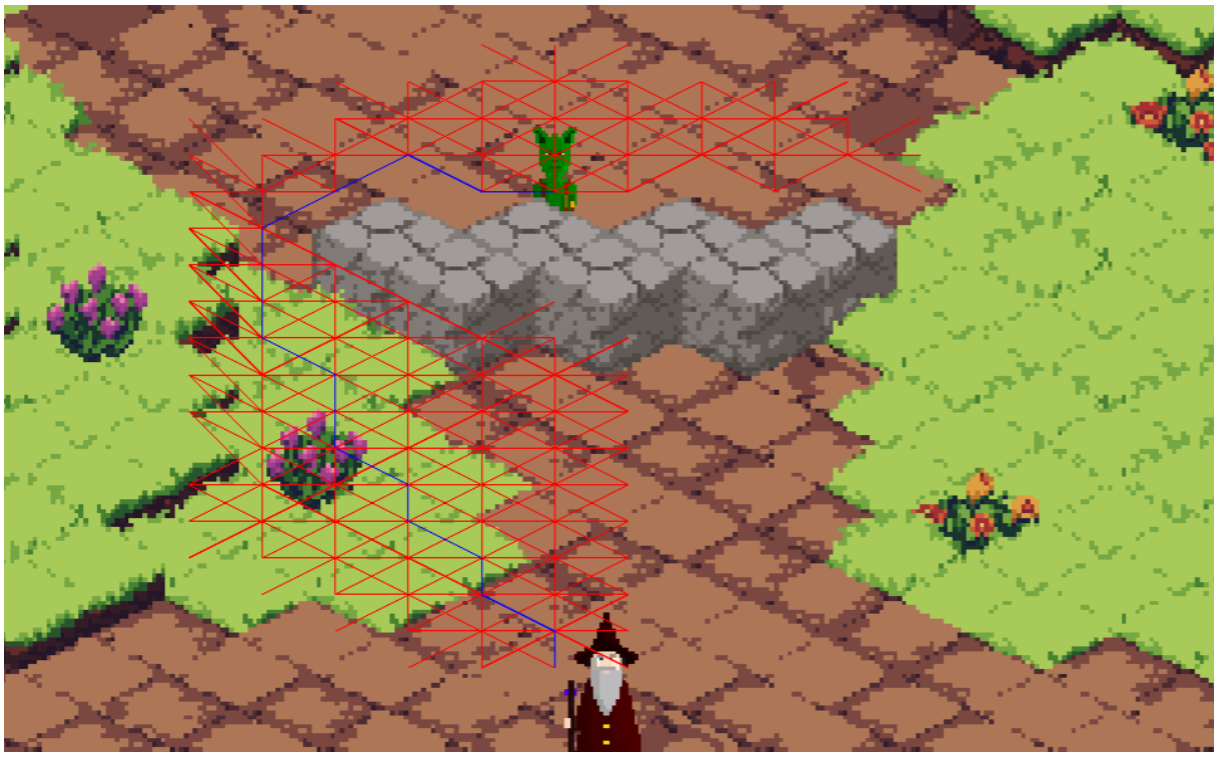

FIGURE 8 – Chemin reconstruit

2.2.2 Optimisation
Antoine, Soutenance 2

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 8. 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 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.

2.2.3 Attaque
Antoine, Soutenance 2

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 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 9) tourne en boucle alors qu'elle pourrait être synchronisée avec les dégâts infligés au joueur.

2.2.4 Monstre à distance
Antoine, Baptiste, Soutenance 3

Quand la distance recherchée par un monstre dépasse un certain seuil, en plus d'assez se rapprocher du joueur, il va aussi s'en éloigner en changeant son objectif de recherche. Ainsi, les monstres à distance ne sont pas toujours collés au joueur et le fuient quand il s'approche trop.

Pour les monstres à distance, la fonction Attack() n'applique pas simplement les dégâts au joueur, elle instancie un préfab de projectile doté d'un RigidBody2D auquel elle applique une force pour créer un mouvement dans la direction du joueur. Le préfab, sa vitesse et ses dégâts peuvent être modifiés sur le préfabs de monstre à distance.

2.2.5 Apparition
Antoine, Soutenance 2

FIGURE 9 – Monstre qui attaque

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 11 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.

2.2.6 Création des monstres
Baptiste, Soutenance 3

Une fois que tous les sprites de monstres et que tous les scripts adéquats ont été finalisé, j'ai pu tout rassembler afin de créer les 11 monstres que nous avons actuellement, pour cela j'ai simplement fait des préfabs de chacun qui seront ensuite facilement plaçables sur les tanières de la map.

2.2.7 Les Boss
Baptiste, Soutenance 3

La classe Boss hérite de Monstre, elle utilise donc la même IA, les 2 seules principaux aspects qui changent sont le fait qu'ils aient une distance maximum à laquelle ils peuvent se rendre par rapport à leur point d'apparition et que leur fonction Attack() est différente. La fonction Attack() des boss instancie des préfabs d'attaque, cela permet d'avoir des attaques plus visuelles et surtout esquivables, ces préfabs sont orientés de manière à donner l'impression qu'ils sont plats mais au dessus du sol bien que le jeu soit en 2 dimensions. Ceux-ci pour infliger des dégâts

FIGURE 10 – Gobelins lançant des clés à molettes

aux joueurs sont simplement équipés d'un script qui gère leur animation, leurs dégats etc... Cela permet de créer les attaques à part du boss pour ensuite pouvoir simplement les glisser dans la liste d'attaques du préfab du boss.

Pour le boss final, il y a quelques particularités telles que le fait de se téléporter lorsque l'on se rapproche trop de lui où encore le fait que pour le combat le joueur doit se téléporter dans un tout autre lieu grâce à ce qu'il aura récupéré sur les 3 premiers boss.

Leurs créations a donc été plutôt aisée après avoir écrit tous les scripts car il suffisait simplement de créer des préfabs.

3 Combat

Le joueur choisit en premier lieu sur son île personnelle un maximum de cinq sorts qu'il a confectionnés pour les mettre dans son inventaire de sorts. Les sorts sont un aspect essentiel du jeu, en effet ils sont le seul moyen dont dispose le joueur pour faire des dégâts aux ennemis.

Dans Hermita, (comme mentionné précédemment) il y a quatre éléments, chaque sort et la plupart des ennemis sont liés à un ou plusieurs de ses derniers (même si certains seront neutres). Ainsi, les sorts feront plus ou moins de dégâts en fonction de l'élément de l'ennemi touché. En combat, il y a également les potions qui sont cruciales notamment en ce qui concerne la gestion de la vie et du mana du joueur, elles peuvent être craftées à l'atelier ou bien lootées sur les *mobs*. Les différents *items* que l'on peut fabriquer existent en plusieurs catégories (la tab. 3 résume leurs différents types et la manière de les produire).

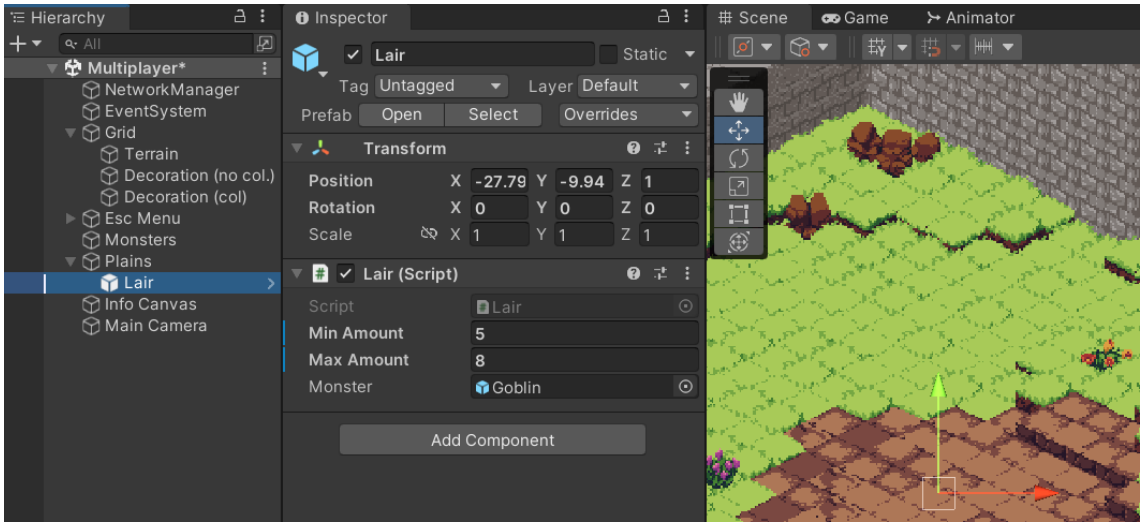

FIGURE 11 – Création d'une tanière dans *Unity*

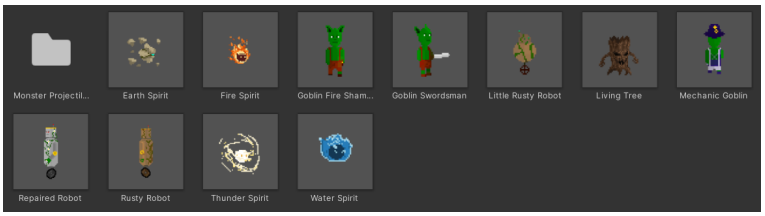

FIGURE 12 – liste des boss

3.1 Statistiques

3.1.1 Classe mère
Antoine, Soutenance 1

Pour gérer ces statistiques, j'ai créé une classe Stats qui est un dictionnaire permettant de stocker des flottants associés à des chaînes de caractères tout en connaissant le type de statisque. En effet, il faut différencier hp +50 et hp +50% par exemple. Cela permet de faire des calculs plus complexes sur les statistiques tout en gardant une certaine flexibilité. Ainsi, chaque objet qui interagit avec les statistiques est une instance de Stats : equipment, consumable, player, etc. J'ai aussi implémenté une classe FlatStats assez similaire mais sans la notion de pourcentage. En effet, dans la base de donnée on peut avoir un équipement qui donne +10% de vie (hp) mais pour optimiser les calculs, +10% est « applati »en fonction de la vie du joueur. Pour ces calculs j'ai défini les opérateurs + et - et d'autres fonctions telles que Flatten et Ceil.

3.1.2 Système d'expérience
Antoine, Soutenance 3

N'avoir que des objets comme matière de progression serait décevant. J'ai donc ajouté la possibilité d'augmenter de niveau en gagnant de l'expérience. Chaque niveau donne 3 points de compétence qui servent à augmenter les statistiques du joueur et chaque amélioration de compétence coûte de plus en plus cher mais donne toujours 5% de bonus. Vous pouvez voir dans la Figure 14 l'interface des compétences. Le système de niveaux est similaire à beaucoup

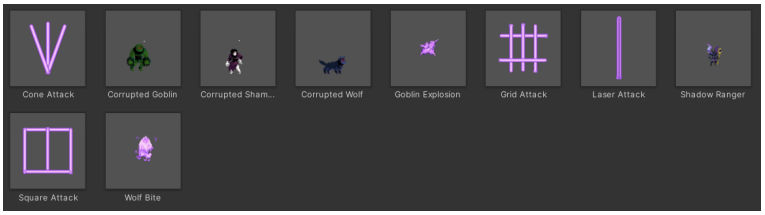

FIGURE 13 – liste des boss

FIGURE 14 – Compétences

FIGURE 15 – Niveaux

de jeux, plus le niveau est élevé, plus il faut d'expérience pour passer au niveau suivant. Et on peut retrouver les informations sur le niveau et le nombre de points de compétence dans l'inventaire (voir Figure 15). Pour simplifier les calculs, les améliorations de compétences sont prises en compte comme si elle constituaient une autre pièce d'équipement. Ainsi, les statistiques du joueur sont recalculées à chaque fois qu'il change d'équipement ou qu'il améliore une compétence. Ce qui est particulièrement utile pour le multijoueur puisque les deux peuvent augmenter les points de vie du joueur et le serveur doit être mis au courant de ces changements. On peut donc regrouper cet appel dans l'événement du changement d'équipement. Finalement, si le joueur veut essayer d'autres statistiques, il peut réinitialiser ses compétences pour récupérer tous ses points de compétence et les répartir à nouveau.

3.2 Multijoueur

Afin d'implémenter le multijoueur, même si l'hébergement de notre jeu ne nécessite pas de serveur, on ne peut pas connecter directement des ordinateurs sans qu'il soient en réseau local dans la plupart des cas et nous ne pouvons pas nous permettre de demander aux joueurs d'ouvrir les ports de leur routeur. Nous avons donc besoin d'un relai entre le joueur qui héberge la partie et les autres.

Pour répondre à cette problématique, nous avons décidé de ne pas opter pour Photon mais plutôt d'utiliser Unity Relay qui permet (avec la version gratuite) d'avoir jusqu'à 50 joueurs mis en relation en simultané. Cette limite pourra être repoussée lorsque Epistars™ entamera une levée de fonds. Pour simplifier l'implémentation du multijoueur dans Unity, nous avons opté pour la solution Netcode qui est principalement destinée à faciliter la création de jeux coopératifs à petite échelle. Cette bibliothèque prend en charge des éléments tels que la gestion des scènes, l'animation ou encore la physique.

L'association des outils Netcode et Relay correspond parfaitement à nos besoins puisque le

jeu « Boss Room » pris en exemple par Unity est similaire en termes de fonctionnalités réseaux et utilise les mêmes outils.

3.2.1 Héberger et Rejoindre
Antoine, Soutenance 1

Le multijoueur possède les fonctionnalités pour héberger et rejoindre une partie. Le travail a été plutôt simplifié par Unity Relay puisqu'il existe nativement un système de parties protégées par un code, exactement ce qu'il nous faut. La joie était présente lorsque ça a marché pour la première fois, mon personnage bougeait en temps réel chez mon colocataire Baptiste ; dont le personnage (un carré) bougeait en temps réel chez moi et vice-versa !

Avant d'arriver à cela, le debug était pénible. Unity ne permet de lancer 2 instances du jeu dans l'éditeur... Solution : faire un build à chaque modification pour lancer le .exe. Heureusement, beaucoup de choses se simplifient grâce à l'absence de sécurité. Aucune vérification de l'identité du joueur, pas de vérification de la validité des données reçues, le paradis des tricheurs ! On part du principe que les codes de parties ne sont partagés qu'entre amis, et que les amis ça se fait confiance.

Cependant, héberger et rejoindre n'est que le début du projet, la plupart du travail sur le multijoueur sera sûrement de gérer tous les bugs et les cas particuliers. Ces problèmes ont déjà été imaginés dès le début de notre aventure, c'est pourquoi il nous semblait important que le système du multijoueur soit implémenté en amont de tout le reste afin que les fonctionnalités se construisent dessus et non l'inverse. Pour pouvoir tester j'ai implémenté 2 simples scripts de mouvement avec les animations non correctement intégrées et j'ai eu un aperçu de la difficulté...

3.2.2 Synchronisation des monstres
Antoine, Soutenance 2

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 16, 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 16 – 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.

3.3 Les sorts

3.3.1 La classe Spell
Baptiste, Soutenance 3

Le moyen le plus simple de gérer les sorts pour ne pas avoir à faire un inventaire de sorts qui aurait presque le même fonctionnement que notre inventaire d'objets a été de faire les sorts comme des items. J'ai donc créé la classe Spell comme héritière de item ce qui permet de stocker les sorts dans la base de données d'items, de plus celle-ci stocke également le Type de sort (projectile, laser ...), le coût en mana, le délai de récupération, le chemin d'accès au préfab du sort, le tier du sort (1, 2 ou 3) et l'élément du sort (feu, eau, terre, foudre).

3.3.2 Les projectiles
Baptiste, Soutenance 3

Le premier type de sort est le projectile, celui-ci est simplement un sprite envoyé dans une direction qui lorsqu'il rentre en collision avec un ennemis (ou un allié pour les sorts de soin) ou un mur ou qu'il arrive au bout de sa portée instancie son préfab d'explosion qui joue l'animation d'explosion et qui inflige les dégâts aux entités dans son collider.

Le sort s'envoie dans la direction de la souris grâce à une force appliquée au rigidbody, il détecte si il touche un mur grâce à la fonction tilemap.HasTile(). Grâce à cela, il suffit de créer des prefabs animés et de leur associer le script ainsi qu'un préfab d'explosion et un sort projectile est créé.

J'ai également implémenté 2 variantes du projectile, la mine qui elle reste sur place et s'active lorsqu'un ennemi (ou un allié si c'est une mine de soin) marche dessus. Ainsi que le TranslationProjectile qui lui inflige des dégâts sur toute sa trajectoire et n'explose pas au contact d'un ennemi.

3.3.3 Les lasers
Baptiste, Soutenance 3

Les lasers constituent des préfabs composés de plusieurs parties de laser, ce découpage permet de détecter si le laser est dans un mur et si oui de le couper à l'endroit de la partie dans le mur. De plus le laser n'est pas seulement instancié et envoyé comme le projectile, celui-ci reste un certain temps entre le joueur et son curseur, ce qui rend le sort technique, mais satisfaisant à utiliser. Les dégâts du laser sont infligés régulièrement (délai défini dans le préfab) sur le

FIGURE 17 – exemple de sort Projectile

compositeCollider qui regroupe les colliders de chaque parties, ce qui permet d'avoir un collider de la même taille que le laser.

FIGURE 18 – exemple de sort laser

3.3.4 Les effets
Baptiste, Soutenance 3

Le dernier type de sorts est l'effet, ceux-ci sont aussi des préfabs car cela rend la chose plus modulable et permet d'afficher une animation d'effet sur le player, ces sorts appliquent simplement un effet sur le joueur tout en jouant un sprite animé qui reste par dessus le joueur.

FIGURE 19 – exemple de sort Effet

3.3.5 Les lancements de sorts
Baptiste, Soutenance 3

Le lancement des sorts se fait en plusieurs étapes, la première se fait dans la barre de sorts c'est là qu'est détécté la pression de touche correspondante à l'un des slots de sort, là les critères de lancement du sort comme le coût en mana ou le temps de recharge du sort sont vérifiés. A savoir que chaque sort a un temps de recharge qui lui est attribué mais il y a aussi un temps de recharge global de 0.5 secondes qui empêche peu importe les statistiques du joueur ou les sorts qu'il lance de lancer 2 sorts à moins de 2 secondes d'intervalle. Puis la fonction CastSpell() du player est appelée celle-ci instancie le préfab dans la bonne direction et enfin qui appelle la fonction ApplyToPrefab() de SpellCast qui elle va appliquer la force adéquate en fonction du type de sort.

3.3.6 Les sorts en multijoueur
Antoine, Soutenance 3

J'ai fait face à une difficulté de taille, si le joueur améliore ses statistiques, il faut que les sorts soient améliorés aussi. Malheureusement, le serveur n'est pas mis au courant des statistiques du joueur, lorsqu'elles sont modifiées. Cette réalisation s'est suivie par un long et pénible debuggage. Pour éclaircir le problème, on doit connaitre la puissance du joueur pour calculer les dégâts des sorts, sa défense pour calculer les dégâts reçus et sa vie pour savoir sur quoi appliquer les dégâts. Si le joueur change ses statistiques, le serveur ne connaissait aucune de ces valeurs. De plus, les statistiques du joueur sont stockées dans un dictionnaire, ce qui rendrait la tâche encore plus difficile si on voulait simplement tout synchroniser car ce serait peu optimisé. Après avoir passé des heures à jouer au serpent qui se mord la queue,

j'ai finalement trouvé un équilibre. Quand un joueur subit l'effet d'un sort qui change ses statistiques, le serveur et le client du joueur seulement sont mis au courant de ce changement. Quand un joueur augmente sa vie via de l'équipement ou des améliorations, le serveur est mis au courant de ce changement et met à jour la vie ainsi que la vie maximale du joueur. Vie et vie maximale sont les seules statistiques qui sont synchronisées via des `NetworkVariables` chez tous les joueurs pour bien afficher la vie des autres joueurs. Finalement, quand un joueur lance un sort, il transmet au serveur sa puissance pour que les bonnes valeurs de dégâts soient calculées. J'ai finalement retrouvé la paix intérieure, un peu trop tard dans la nuit, quand je suis arrivé à faire fonctionner tout ça comme dans la Figure 20.

FIGURE 20 – Les sorts synchronisés

3.3.7 Création des sorts
Baptiste, Soutenance 3

Avec tous les scripts créés pour les différents types de sorts auxquels on avait pensé, créer les différents sorts consistait seulement à créer leurs préfabs avec les bonnes animations et les bonnes valeurs, ainsi que de créer l'instance de Spell() dans la base de données des items et le tour était joué. Nous nous retrouvons donc avec 14 spells au final qui constituent les 14 combinaisons possibles de 1 ou 2 éléments, et cela permet d'avoir beaucoup de variété dans le gameplay du joueur.

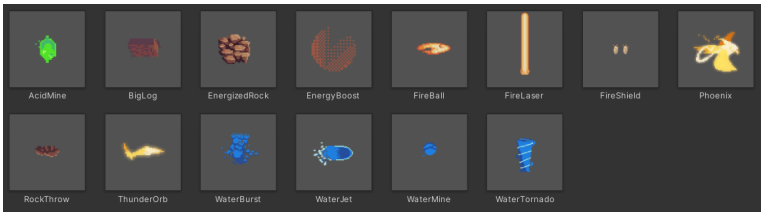

FIGURE 21 – liste des sorts

4 Art

Le projet Hermita fonde sa direction artistique sur la dualité entre la magie considérée comme naturelle dans le monde et l'aspect artificiel des machines *steampunk* amenées par le joueur lorsqu'il vient perturber ce monde a priori bien réglé.

4.1 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 Bussière 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.

4.2 Graphisme

Le design du jeu sera conçu en pixel art mais sous forme isométrique**?** (la fig. 8 indique nos références) exception faite de la cinématique de départ qui sera, quant à elle, réalisée dans un style 2D plus « classique ». Le style général marquera l'opposition entre la nature emplie de magie des environnements et les machines (dans un style *steampunk***?**) pour les autres éléments.

4.2.1 Procreate
Dylan, Soutenance 1

Utilisant Procreate depuis plusieurs années, j'ai décidé d'utiliser ce logiciel pour réaliser le graphisme de notre jeu, puisqu'il permet non seulement de dessiner les différents assets, mais aussi de réaliser les animations. Cependant, celui-ci n'est pas conçu pour réalisé des dessins en pixel art dans sa version de base. J'ai donc, grâce à de nombreux tutoriels que l'on peut aisément trouver sur internet, créés des pinceaux permettant la réalisation de ces dessins.

4.2.2 Logo
Dylan, Soutenance 1

Le logo, qui est l'une des premières choses que l'on voit sur un jeu, a deux buts principaux. Le premier est de commencer à raconter l'histoire du jeu avant même de débuter la partie. Pour rappelle, le mage s'est écrasé sur l'île avec son dirigeable. Ainsi, quoi de mieux que de représenter un dirigeable en guise de logo ! Le second était de montrer le style de notre jeu : le pixel art. Ainsi, dans cette idée de représentation du style, le logo Hermita est passé par une certaine phase d'évolution (Figure 22), voyant ainsi le nombre de pixels réduire au fil des versions.

A noter que les couleurs utilisés pour ce logo sont similaires à celles du mage.

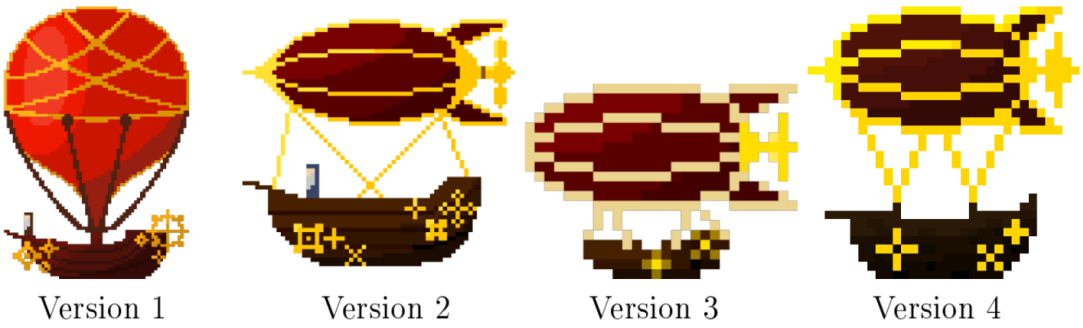

FIGURE 22 – Evolution du logo Hermita

4.2.3 Le mage
Dylan, Soutenance 1

Ne maitrisant pas entièrement les codes du pixel art, le processus de création du mage a été un peu long au début. Je suis donc passé par une phase de dessin plus réaliste, puisque c'est un genre auquel je suis plus habitué, comme on peut le voir dans la Figure 23.

C'est seulement par la suite, après de nombreuses tentatives, que j'ai réussi à créer la toute première image en pixel art de notre mage (Figure **??**).

Cette première étape m'a beaucoup appris sur la manière de dessiner en pixel art, un monde totalement nouveau si l'on compare cela aux autres genres de dessin. Cela m'a aussi appris à travailler les ombres d'une manière différente. Ces nouvelles méthodes m'ont permis par la suite de débuter la réalisation des différentes animations, afin que le mage puisse exécuter différentes actions dans le jeu. La première animation était la suivante : le faire marcher. Dans une telle animation, la diffculté réside surtout dans le fait que chaque image soit cohérente avec la suivante. Sachant que l'animation doit se répéter indéfiniment de façon cyclique, il faut aussi que la dernière image de l'animation soit cohérente avec la toute première. Le but étant que le mouvement soit fluide et qu'il n'y ait pas de décalages flagrants entre 2 images consécutives. (Figure 25)

Nous avons décider que le mage pourrait avancer sur 8 côtés, il m'en restait donc 7 à réaliser. Ici, de nouvelles diffcultés sont apparues. Pour pouvoir réaliser le personnage sur plusieurs côtés, même s'il est en 2D, il faut pouvoir se le représenter en 3D pour ensuite le projeter dans un monde 2D. Les questions à se poser à ce moment présent pour chaque sens de marche du personnage sont les suivantes : Qu'est-ce qui doit être visible ? Ou caché ? Est-ce que les proportions corrèlent entre chaque axe de mouvement ? Les ombres sont-elles bien placées ? Après un certain moment j'ai pu aboutir et obtenir les 8 directions, comme on peut le voir sur la Figure 26 :

Un mage qui marche, c'est bien ... Mais le voir courir, c'est encore mieux. En effet, un personnage qui ne ferait que marcher en jeu rendrait le tout un peu long et ennuyant. C'est pourquoi il fallait que notre mage puisse aller plus vite en courant. Sachant que le gobelin avait lui aussi été animé en train de courir, je savais déjà comment m'y prendre pour le mage. De plus, en ajoutant du mouvement à la cape, j'ai pu rendre le mouvement plus réaliste, et apporter un effet de rapidité. Le résultat sur la Figure **??**

FIGURE 23 – croquis du mage

FIGURE 24 – mage en pixel art

FIGURE 25 – Animation du mage marchant vers la droite

4.2.4 Le gobelin : un premier ennemi ! Dylan, Soutenance 1

Ayant beaucoup appris avec la création du mage, ce nouveau personnage m'a semblé au départ plus facile à appréhender. Même si un détail change un peu la donne. En effet, sur le mage, la cape venait cacher ses jambes, je n'avais donc pas tellement à prendre en compte le mouvement des jambes. Mais ici, c'est différent, puisqu'elles sont bien visibles. C'est en raison de ce détail que cela a mis un certain temps à réaliser cette animation, mais j'ai fini par y arriver. Par ailleurs, pour dessiner le gobelin, je me suis inspiré de ceux présents dans Clash of Clans. Le résultat sur la figure 28

4.2.5 Le mage à l'attaque Dylan, Soutenance 2

FIGURE 26 – animation du mage (marche)

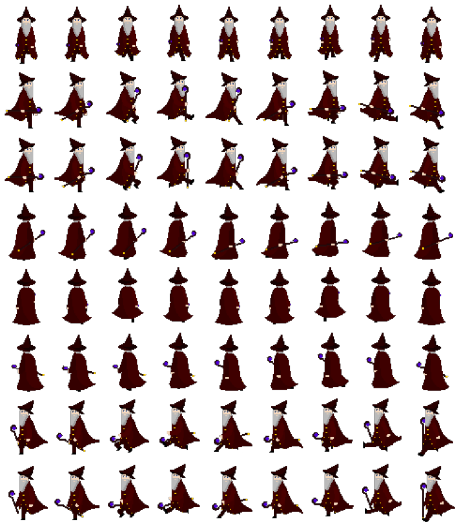

FIGURE 27 – animation du mage (course)

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 Figure 29. Aussi, j'ai apporté au niveau de la boule au 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 30.

Ainsi j'ai réalisé deux types d'attaques, sur les huit côtés que l'on peut voir dans les Figure 31 et Figure 32.

4.2.6 Un autre gobelin
Dylan, Soutenance 2

Avec l'équipe, nous avons prévu de réaliser au moins deux ennemis principaux, c'est-à-dire un gobelin et un robot. Mais nous avons aussi dans l'idée que ces deux types d'ennemis soit en réalité les racines de deux 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 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 33.

4.2.7 Les nouveaux ennemis
Dylan, Soutenance 3

Durant tout ce projet, j'aurais réussi à créer 6 ennemis au total. Nous avons donc : le gobelin

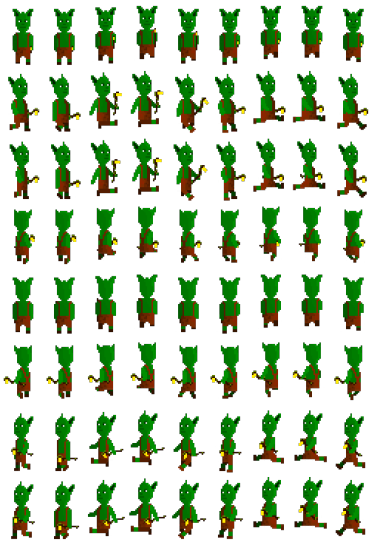

FIGURE 28 – Animation du gobelin

chaman (Figure 34), le gobelin épéiste (Figure 35), le gobelin mécanicien (Figure 36), le robot réparé (Figure 37), le robot rouillé (Figure 38), et enfin le petit robot rouillé (Figure 39).

4.2.8 Camps de gobelins
Dylan, Soutenance 3

Certains dessins pour les camps de gobelins ont été réalisés (tentes, bannières, feu de camp, torches, coffres, etc.), le but étant de rendre la carte plus vivante en jeu. On peu retrouver un des camps de gobelins dans la Figure 49

4.2.9 Les objets
Baptiste, Soutenance 2

Comme j'avais pour rôle de créer et d'implémenter la liste des items, je me suis dis que ce serait plus pertinent des les dessiner moi même, pour cela j'ai utilisé le logiciel piskel. J'ai donc dessiné les Potions, les composants élémentaires pour les sorts, les gemmes, les colliers et tous les composants de craft. Par la suite, quand j'ai dû ajouter de nouveaux objets au jeu, je me suis fait aider par deux amis à mois, Thomas et Samira, qui ont dessiné respectivement les équipements et les parchemins de sorts.

4.2.10 Assets de terrain
Baptiste, Soutenance 2

Nous avions deja plusieurs assets de terrain mais ceux-ci constituaient principalement des éléments de décors naturels et nous avions besoin d'asset pour construire des choses fabriquées par l'homme. J'ai donc dessiné des tiles de pierre avec différentes variations qui nous ont permis de construire ensuite des bâtiments et des ruines sur notre carte.

FIGURE 29 – Attaque 1

FIGURE 30 – Attaque 2

4.2.11 Assets achetés
Baptiste, Soutenance 2

Comme faire un RPG d'aventure demande énormément de dessins différents, nos savions d'entrée qu'il ne serait pas possible de tout dessiner par nous même, nous avons donc pris la décision d'acheter les assets qui constituent les éléments basiques d'un jeu vidéo (comme les assets de terrain) et de dessiner nous même ceux qui sont plus essentiels à notre jeu (personnages, items etc...).

Je me suis donc occupé de chercher les assets pour tout ce qu'il nous manquait, que ce soit en achetant des assets adéquats ou en modifiant certains pour qu'ils correspondent à nos besoins.

Tous les sprites de boss et une partie des assets de terrain, des monstres et des effets de sorts on été achetés, la plupart ont par la suite été modifié pour coller aux couleurs et au thème de notre jeu. Au final plus de la moitié de nos assets ont été dessinés par nos soins ce qui est un ratio raisonnable au vu de la quantité de dessins nécessaires à notre jeu.

4.3 Level Design

4.3.1 L'île d'housing
Baptiste, Soutenance 1

La première étape du level design a été de construire l'île du housing, en effet nous avions besoin d'un endroit où commencer à développer le jeu en tant que tel. Pour cela j'ai donc créé une petite île sur laquelle j'ai placé les différents bâtiments le tout avec une décoration très naturelle assez simpliste.

4.3.2 Les bases de l'île multijoueur
Baptiste, Soutenance 1

FIGURE 31 – Animation attaque 1

FIGURE 32 – Animation attaque 2

Après avoir créé la petite île du solo, il a fallu se lancer dans le gros morceau, l'île multijoueur, pour celle-ci je me suis occupé de créé les bases de l'île, principalement la forme, les reliefs, les routes, les différentes zones et leurs thèmes. Grâce à cela les bases étaient posées pour que tout le monde puisse contribuer à la création de cette carte.

4.3.3 La zone de départ
Baptiste, Soutenance 1

Cette zone constitue le premier aperçu que le joueur se fait de l'aventure dans notre jeu, il fallait donc quelque chose de plutôt accuillant et dans le même thème que l'île du housing. En parallèle de l'île de départ vient aussi l'île du combat de boss final, celle-ci est une simple plateforme circulaire car c'est ce qui permet de créer un combat le plus fluide possible.

4.3.4 Les ruines
Dylan, Soutenance 3

Pour continuer à développer la carte du jeu, je me suis occupé dans un premier temps des ruines. Des vestiges de château ont ainsi été ajoutés, fabriqués à l'aide des assets de pierres que nous avions déjà. De nombreux camps de gobelins ont aussi été ajoutés. En ces lieux se concentrent de nombreux ennemies. Cela nous permet d'avoir une carte enrichie, avec de nombreuses nouvelles zones à découvrir ! On peut voir à quoi ressemble cette zone sur les Figure 46 et 47.

4.3.5 La plaine
Dylan, Soutenance 3

Ensuite, je me suis occupé de la plaine. Ici rien de plus simple : nous ajoutons quelques

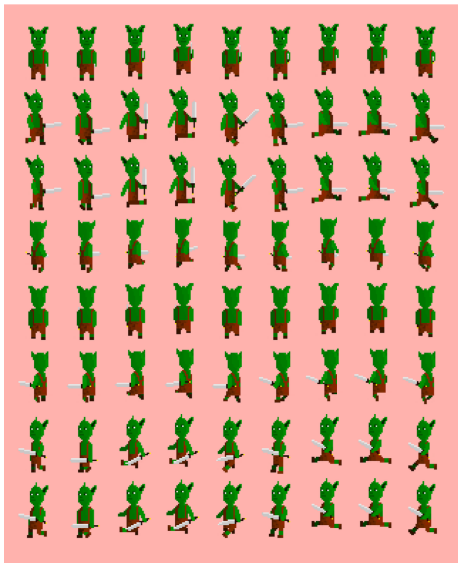

FIGURE 33 – Animation du gobelin épéiste

arbres par ici, quelques buissons et fleurs par là, et le tour est joué ! Tout comme les ruines, la plaine regroupe quelques camps de gobelins ! Il y a aussi un plan d'eau, où de nombreux ennemis vous attendent ! Une partie de la plaine sur la figure 48.

4.3.6 La forêt
Antoine, Soutenance 3

Pour la dernière zone, il nous fallait une nouvelle thématique pour contraster avec la plaine et les ruines, pour cette raison, nous avons opté pour implémenter une forêt. La principale problématique a été les arbres, en effet nous ne voulions pas des arbres trop gros qui cacheraient ce qu'il se passe à l'écran, mais pas trop petit pour que ça ait de la cohérence. J'ai également assombri les textures de l'herbe et de la terre pour donner un aspect sombre à la zone et ajouté d'autres petits assets tels que les champignons.

4.3.7 Les camps de gobelins
Dylan, Soutenance 3

Certains dessins pour les camps de gobelins ont été préalablement réalisés (tentes, bannières, feu de camp, torches, coffres, etc.). Ainsi j'ai pu implémenter cela dans le jeu pour obtenir de magnifique camps de gobelins. Ces-derniers ont deux buts : le premier est de rendre la carte plus vivante. C'est bien d'avoir des ruines, une plaine, une forêt ... Mais c'est encore mieux si l'on peut montrer que ce monde est peuplé d'étranges créatures ! Le deuxième but notable est de faire comprendre que cette zone est plus à risque qu'une autre car il y a de nombreux ennemis présents. Notons que nous avons deux types de camps de gobelins. Le premier type est celui qui a été ajouté aux ruines et aux plaines. C'est un camp assez clair, qui correspond bien à ces deux zones. Le second type est plus sombre, pour correspondre aux couleurs de la forêt. On peu retrouver deux camps de gobelins dans la Figure 49 et la Figure 49

FIGURE 34 – Gobelin chaman

FIGURE 35 – Gobelin épéiste

FIGURE 36 – Gobelin mécanicien

4.3.8 Système de zones
Baptiste, soutenance 3

Le système de zone a de base été implémenté pour pouvoir faire un système de respawn cohérent. Pour découper la map j'ai donc créé des edgeColliders autour de chaque zone afin de pouvoir détecter les changements de zone. A chaque changement de zone plusieurs choses s'actualisent, comme le point de réapparition, les musiques ainsi que l'apparition d'une bannière sur l'interface qui indique le changement de zone.

4.3.9 Placement des monstres
Baptiste, soutenance 3

Une fois que la map était totalement construite, il a fallu que je me lance dans le placement des monstres sur celle-ci. Ayant déjà créé tous les préfabs de monstres, Ceux-ci étaient déjà créé de manière à correspondre plus ou moins à une zone en particulier. Je les ai donc placé dans leur zone en utilisant les lairs créés par antoine, de manière à ce que la difficulté vienne de manière croissante.

Après avoir placé les mobs, il a fallu également placer les boss, pour cela j'ai utilisé le même procédé que pour les monstres, car les propriétés des lairs convenaient également à une bonne implémentation de boss. Au final, il y a donc 1 miniboss par zone et le boss final se trouve sur sa propre île accessible en tuant les 3 autres.

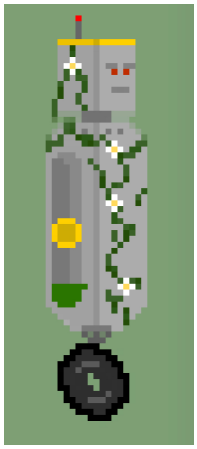

FIGURE 37 – Robot réparé

FIGURE 38 – Robot rouillé

FIGURE 39 – Petit robot rouillé

5 Objets

5.1 Organisation

5.1.1 Classes mères
Antoine, Soutenance 1

Les items sont découpés en 3 classes : Item, Equipment et Consumable. Item est la classe mère de ces deux dernières. Elle contient les propriétés communes à tous les items : nom, description, icône, etc. Equipment et Consumable contiennent les propriétés spécifiques à ces deux types d'items. Par exemple, Equipment contient les propriétés de l'équipement (armure, dégâts, etc.) et Consumable contient les propriétés du consommable (effet, durée, etc.).

J'ai aussi implémenté des recettes pour avoir un squelette de base mais ce sera une tâche à finir plus tard.

5.1.2 Bases de données
Antoine, Soutenance 1

Pour ajouter des items ou des recettes, il suffit de modifier les fichiers qui agissent comme des bases de donnée. Ces bases de données renvoient l'objet associé à son id. Chaque item (ou recette) n'est instancié qu'une fois puisque ses propriétés ne changent pas. Ce n'est juste pas le cas pour l'équipement dont une nouvelle instance est créée dès que l'objet est récupéré dans la base de données puisque ses stats subissent un passage à l'aléatoire. Ces scripts sont

FIGURE 40 – Camp de gobelin

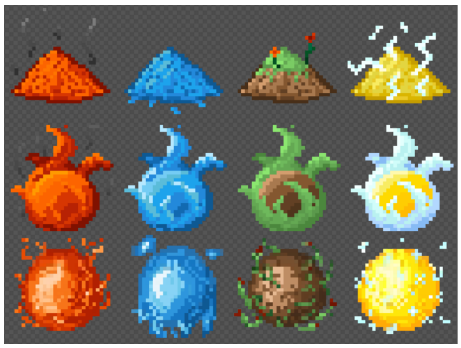

FIGURE 41 – Dessins de objets élémentaires

rattachés à un GameObject dédié aux bases de données. Les constructeurs d'Item et de Stats sont surchargés et simplifiés (voir exemple ci-dessous) pour rendre la tâche du game designer plus aisée.

5.1.3 Le joueur
Antoine, Soutenance 1

Pour associer un réel objet à toutes ces statistiques, j'ai créé la classe Player qui permet d'avoir des statistiques de base, un équipement et des effets. Je gère aussi une boucle de régénération pour que toute statistique de la forme *-regen soit une valeur de régénération par seconde. C'est ici que sont définies les fonction Equip pour l'équipement et UseConsumable pour les potions.

5.1.4 GameDesign Items
Baptiste, Soutenance 1

Dès le début du projet j'ai rédigé une liste de tous les items qui seraient obtenables dans le jeu : les bâtons, les robes, les colliers, les potions, les loots élémentaires, les gemmes et les composants d'artisanat. Chaque item y est détaillé avec son utilité, sa rareté, son obtention ainsi qu'une illustration de ma vision graphique de cet item. Par la suite, cette étape, combinée au travail déjà effectué par Antoine, a rendu très facile le remplissage de la base de données d'items.

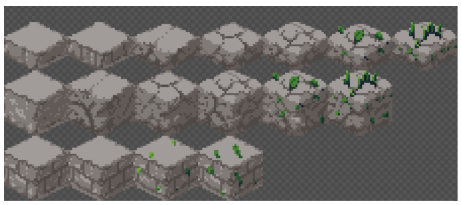

FIGURE 42 – Dessins de tiles de pierre

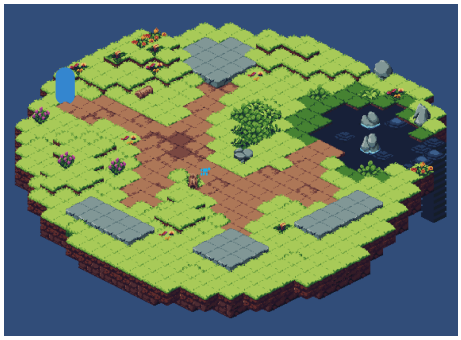

FIGURE 43 – Ile du housing

5.2 Inventaire

5.2.1 Début du système
Antoine, Soutenance 1

Bien que la tâche ne devait être que commencée, j'ai tout de même voulu rendre l'inventaire du joueur fonctionnel pour pouvoir implémenter les vraies fonctionnalités par la suite. Les joueurs n'ont pas de coffre, il est donc important de leur fournir un inventaire complet en fonctionnalités. C'est pourquoi déjà maintenant, on peut choisir de montrer "tout", "équipement", "consommables" et "ressources". En plus de ce filtre, on peut trier par ordre alphabétique, par rareté et par quantité, chacun dans l'ordre croissant ou décroissant. L'inventaire est infini et n'est donc pas constitué de *slots* comme dans la plupart des jeux mais simplement d'une liste chaînée. Dans l'UI, il a ainsi fallu mettre les items dans quelque chose où l'on puisse scroller de manière théoriquement infinie.

Les items sont représentés dans l'interface par des instances de la classe UIItem qui instancie un prefab dans l'inventaire. Pour interagir avec eux, on peut passer la souris dessus ce qui affiche un petit encadré avec plus d'informations sur l'item et éventuellement ses statistiques. En faisant un clic droit sur un équipement on peut l'équiper et en faisant un clic droit sur une potion on peut l'utiliser. Les statistiques du joueur sont affichées sur la gauche et actualisées grâce à un évènement déclenché dans la classe Player.

5.2.2 Favoris et menu contextuel
Antoine, Soutenance 2

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 53.

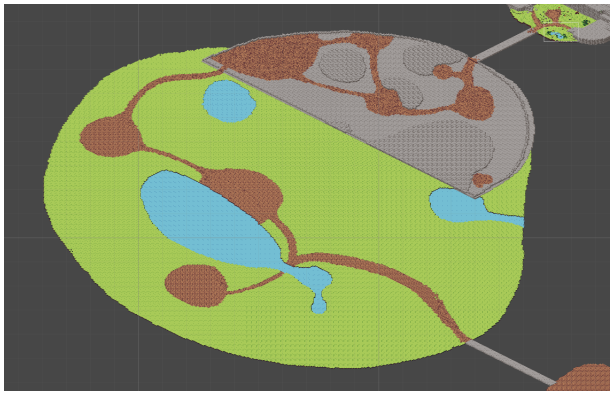

FIGURE 44 – Bases de la deuxième île

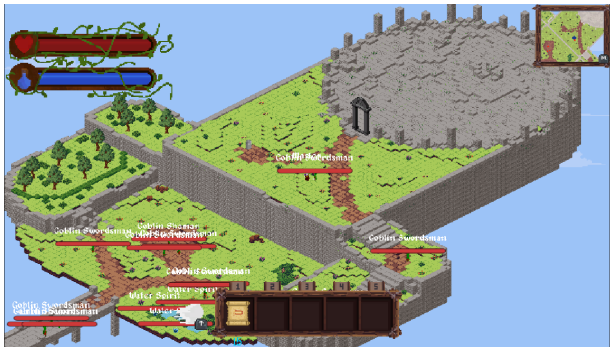

FIGURE 45 – Ile de départ du multijoueur

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 53)

5.2.3 Rareté
Antoine, Soutenance 2

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 54.

5.2.4 Sauvegarde
Antoine, Soutenance 3

L'inventaire contient toutes les informations les plus importantes du joueur, à savoir : les

FIGURE 46 – Une ruine

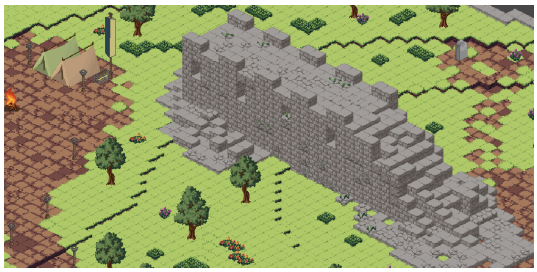

FIGURE 47 – Une autre ruine

objets qu'il possède, les objets qu'il a équipé, son expérience, son niveau, ses compétences (3.1.2) et les sorts équipés. Il est donc important de pouvoir sauvegarder l'inventaire du joueur, sinon il perdrait tout son équipement et ses objets à chaque fois qu'il quitte le jeu. Les objets non-sérialisables, comme les listes, sont transformés en *strings* à l'aide de la fonction `JsonConvert.SerializeObject()`. Cela permet de créer une classe qui vérifie les propriétés de `[Serializable]` et de les sauvegarder dans un fichier à l'aide d'un `BinaryFormatter`. Les fichiers de sauvegarde sont stockés dans le dossier `Application.persistentDataPath` qui est différent selon les systèmes d'exploitation. Sur Windows, il s'agit de `C:/Users/username/AppData/LocalLow/Epistars/Hermita/`. Pour l'inventaire, le fichier utilisié est `Inventory.dat`.

5.3 Crafting

5.3.1 Base de données
Baptiste, Soutenance 1

La première chose qui a été faite avant d'implémenter le système de craft a été la classe Recipe, celle-ci fait le lien entre les ingrédients et le produit final d'un craft. Elle indique aussi le bâtiment dans lequel le craft doit se faire, le type de craft ainsi que le temps de craft. De la même manière que pour la base de données des items, cette classe a permis de répertorier tous les crafts. La tâche de remplir cette base de données a encore une fois été plutôt rapide car cela consistait principalement à recopier les documents de game design déjà établis.

5.3.2 WorkBench
Baptiste, Soutenance 2

La Workbench est le bâtiment qui permet de broyer des géodes pour obtenir une gemme

FIGURE 48 – La plaine

FIGURE 49 – Un camp de gobelins dans les ruines

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. Pour la créer j'ai tout d'abord implémenter un sprite ayant le script Building associé pour créer le bâtiment sur l'île, ce script permet d'intéragir avec le sprite pour ouvrir une interface. Ensuite j'ai donc développé l'interface avec les différents éléments tels que les panels, les boutons, les sliders etc... A tous ces éléments j'ai associé différents scripts, un script pour l'interface en globalité, l'affichage du craft sélectionné, le craft en lui même au niveau de l'inventaire, l'affichage de la progression du craft. Pour le craft dans l'inventaire une méthode CanCraft() était déjà implémenté dans la clase Recipe il ne manquait plus qu'à supprimer les objets de l'inventaire et gérer la mécanique de temps de craft. Un deuxième script gère les préfabs d'items craftables affichés et la sélection du craft et un dernier gère simplement les préfabs d'ingrédients.

5.3.3 WorkShop
Dylan, Soutenance 2

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

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 **??**

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

FIGURE 50 – Un camp de gobelins dans la forêt

```
new Equipment(
    id: 6,
    name: "Bling bling king cape",
    description: "The cape of the goblin king living in the ruins",
    tier: 'C',
    set1[83],
    StuffType.Robe,
    stats: "res +10%, spd +10%, atk +10%"),
```

FIGURE 51 – Constructeur Equipment

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.

5.3.4 Laboratoire
Dylan, Baptiste, Soutenance 3

Pour le laboratoire, le travail s'est fait en deux temps, Dylan a dans un premier temps créé l'interface de craft de sorts initial de la même manière que les autres bâtiments car nous avons fait en sorte que les sorts soient simplement des items. Cependant, après que le groupe ait pris la décision d'attribuer différents Tiers aux sorts, il a fallu le remodeler pour qu'il convienne aux 3 tiers de sorts, Baptiste s'en est donc occupé car c'est lui qui implémentait au même moment des sorts.

6 Interfaces

6.1 Menus interactifs

6.1.1 Console
Antoine, Soutenance 1

Nous utilisons un Asset du Unity Store pour intégrer une console dans le jeu, que l'on peut ouvrir avec la touche entrée. J'ai toujours trouvé les codes de triches ennuyants à retenir alors j'ai alors proposé de rajouter des commandes pour tester des fonctionnalités. De cette manière, nous avons les commandes give_item, use_consumable et set_stat. À l'avenir peut-être une commande de téléportation ou des façons de débloquer des zones.

FIGURE 52 – Inventaire

FIGURE 53 – Menu contextuel et favoris

6.1.2 Menu multijoueur

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 59. 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.

6.1.3 Menu paramètres

Pour pouvoir offrir la meilleure expérience utilisateur, nous avons décidé d'implémenter un menu de paramètres ouvrable avec la touche échap. Celui-ci contient une barre pour régler

FIGURE 54 – Équipements triés par tier de rareté

FIGURE 55 – Interface de la workbench

le volume des musiques ainsi qu'un menu pour configurer ses contrôles. Pour pouvoir régler les touches du joueur, j'ai créé un CustomInputManager qui stocke les valeurs attribué aux différentes actions, celui-ci est ensuite appelé dans les différents scripts.

6.2 Affichage en surimpression (HUD)

6.2.1 Barre de vie des entités
Antoine, Soutenance 2

La barre de vie est simplement un *Slider* d'*Unity* qui est positionné au-dessus de la tête de l'entité. Pour cela, il suffit de récupérer la position de l'entité, 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 62 avec un nom factice en attendant une implémentation du système de sauvegarde.

6.2.2 Gestion d'erreurs
Antoine, Soutenance 2

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

FIGURE 56 – Workshop à l'ouverture

FIGURE 57 – Apparition de la recette dans le craft

FIGURE 58 – Interface du laboratoire

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 63 un exemple de message d'erreur qui s'affiche si le joueur n'a pas entré de mot de passe.

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.

6.2.3 Objets donnés par les monstres (loot) Antoine, Soutenance 3

Les monstres, pour être intéressants, doivent donner des récompenses aux joueurs lorsqu'ils meurent. Le système est relativement simple, quand un monstre meurt, on donne son loot à tous les joueurs présents dans la partie. Pour que le joueur puisse savoir ce qu'il récupère sur les monstres tués et plus généralement, ce qui rentre dans son inventaire, j'ai ajouté un affichage en bas à droite de l'écran. Cela montre les objets récupérés et leur quantité. Quand on se donne tous les sorts par exemple, on obtient cet affichage visible dans la Figure 64.

Les objets restent à l'écran et si de nouveaux s'ajoutent, ils apparaissent en plus dans l'affichage. Si rien n'est récupéré pendant 5 secondes, l'affichage disparaît et recommence à zéro quand un nouvel objet est récupéré.

6.2.4 Minimap : Construction Antoine, Soutenance 3

Le jeu étant bien avancé, ils nous fallait un moyen de se repérer dans le monde et de voir où sont les autres joueurs. La première étape est de réussir à créer une image représentant le monde avec une vue de dessus. Cependant, le jeu est en isométrique, il faut donc faire une projection

FIGURE 59 – Interface en jeu

FIGURE 60 – Interface des paramètres

de l'image pour que les joueurs puissent se repérer. De plus, nous sommes des programmeurs, il n'est pas question de redesiner la carte à la main à chaque fois qu'on la modifie. Il faut donc trouver un moyen de générer automatiquement cette image.

Pour cela, j'ai codé un script qui prend les tilemaps de la carte et les convertit en image. Cette opération n'est effectuée qu'une fois, du côté du développeur, et le résultat est sauvegardé dans un fichier PNG. J'ai donc décidé de générer cette carte avec 1 pixel par tile, ce qui permet de garder l'aspect pixel art du jeu. La première difficulté est d'associer une couleur à chaque tile. Pour cela, j'ai créé un dictionnaire qui associe une couleur à chaque nom de tile. Toutefois, nous sommes toujours des programmeurs, il faut donc automatiser cette étape. On peut donner une liste de sprites à ce script et il va générer automatiquement le dictionnaire en calculant la couleur moyenne de chaque sprite. Une fois le dictionnaire créé, il suffit de parcourir les tilemaps et de remplacer chaque tile par un pixel de la couleur correspondante. L'image générée est visible dans la Figure 65. [1]

6.2.5 Minimap : Affichage
Antoine, Soutenance 3

1. Nous n'acceptons pas les remarques sur la forme de l'île de départ, elle a été réalisée dans une vue isométrique et non de dessus.

FIGURE 61 – Interface des controles

On pourrait se dire que le plus dur est fait, mais il reste encore à afficher cette image dans le jeu. Il faut déjà appliquer une rotation de 45 degrés à la carte pour qu'elle soit dans la même perspective que le reste du jeu. Ensuite, elle est bien trop grande pour être affichée en entier à l'écran. L'objectif est d'avoir un petit carré qui restreint la zone visible de la carte. Pour cela, j'ai créé un masque qui cache tout ce qui est en dehors de ce carré. Autre problème, la carte doit suivre le joueur, il faut donc la déplacer en fonction de sa position. Sauf qu'à cause de l'isométrique, la carte doit être déplacée en diagonale. Pour trouver un solution à ce problème, j'ai utilisé mes connaissances mathématiques acquises à l'EPITA. Il s'agit d'un problème de changement de base, il faut passer de la base cartésienne à la base isométrique. Ce qui peut être fait avec la matrice de passage suivante :

$$\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \tag{1}$$

Le dernier besoin est de pouvoir afficher les entités sur la carte. Pour réaliser cela, j'ai mis à jour le script de la barre de vie des entités pour qu'en plus ils actualisent leur position sur la carte. Pour les ennemis, un carré rouge et pour les joueurs, un carré vert un peu plus grand. Le joueur qui est en train de jouer est affiché en bleu. Le résultat final est visible dans la Figure 66. Pour offrir un plus grand champ de vision aux joueurs, j'ai ajouté la possibilité d'agrandir la carte avec la touche M ce qui la dézoomme aussi et offre une meilleure vue d'ensemble comme dans la Figure 67.

6.2.6 Barres de vie et de mana
Dylan, Soutenance 3

Pour éviter d'avoir de simples rectangles rouge et bleu en guise de barres de vie et de mana, j'ai décidé de les dessiner, et j'ai obtenu ce que l'ont peut voir dans la figure 68

Ces barres sont toutes deux composées de trois parties : d'abord l'image de fond (de couleur bois), l'image de devant, c'est à dire le feuillage ainsi que le symbol représentant la barre (un coeur pour la barre de vie et une potion pour la berre de mana), et enfin une barre rouge ou bleu qui viennent indiquer la quantité de vie ou de mana. Ainsi, pour représenter la quantité de vie restant au joueur, on vient tout simplement contrôler la largeur de la barre en fonction des statistiques de ce dernier.

FIGURE 62 – Barres de vie

FIGURE 63 – Message d'erreur

6.2.7 Barre de sorts
Baptiste, Soutenance 3

La barre de sort est composée de 5 slots de sorts représenté par un array de Spell, ceux-ci peuvent se voir attribuer un sort lorsque le joueur fait un glisser/déposer d'un Spell de son inventaire vers celui-ci. Leur utilité se joue ensuite lorsque le joueur souhaite lancer un sort et qu'il trigger la fonction CastSpell, dans celle-ci, en fonction de la touche pressée par le joueur, le sort correspondant est envoyé.

En plus des icones de sorts, le mana ainsi que les temps de chargements des sorts sont affichés, les temps de chargements sont représentés par l'icone de sort qui s'affiche progressivement au fil du temps en forme d'horloge.

6.2.8 Dash
Antoine, Soutenance 3

Pour rendre le déplacement plus dynamique, j'ai rajouté la possibilité de dasher. Il existait

FIGURE 64 – Loot

déjà un système de dash dans le jeu, mais il ne prenait pas en compte la vitesse du joueur et n'était pas très agréable à utiliser. J'ai donc décidé de le refaire avec un système de chargement de dash. Plus on attend pour charger le dash, plus il sera puissant, et ce, exponentiellement.

6.3 Écrans

6.3.1 Écran de chargement
Antoine, Soutenance 2

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 70) 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*.

7 Communication

7.1 Site web

7.1.1 Choix des librairies
Antoine, Soutenance 1

Le choix du framework svelte a été fait dès le début, principalement parce que c'est mon préféré. De plus, il est très simple à utiliser et à comprendre, ce qui est un plus pour un projet de groupe. Pour la partie serveur, j'aurais bien pris SvelteKit mais je voulais rendre l'écriture

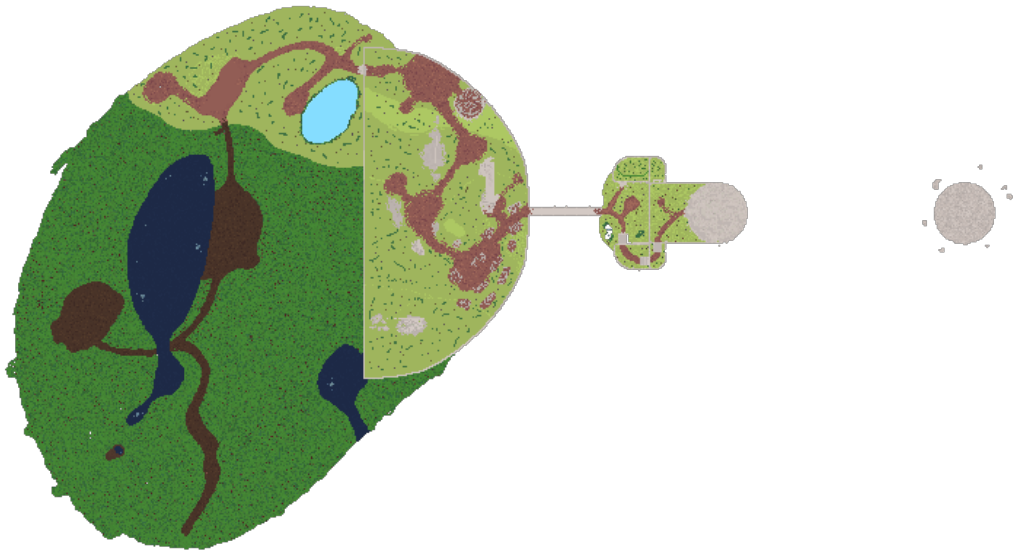

FIGURE 65 – Carte complète[1]

FIGURE 66 – Minimap

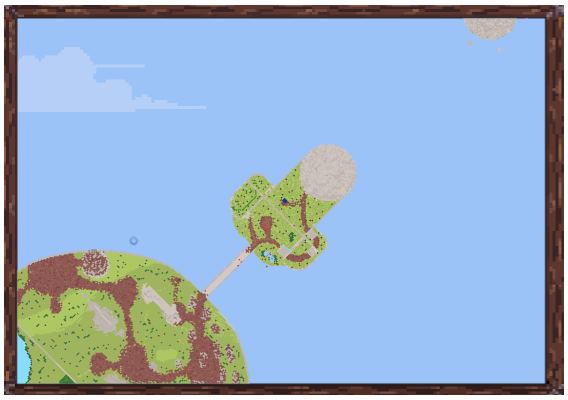

FIGURE 67 – Minimap agrandie

des posts le plus simple possible pour motiver toute l'équipe à en écrire. Astro est alors apparu comme une solution attrayante. C'est un générateur de sites statiques qui permet d'utiliser différents frameworks tels que Svelte ou React pour programmer. Mais une fonctionnalité séduisante consiste à utiliser un dossier de fichiers au format markdown comme une collection, ce qui est adéquat pour implémenter un système de blog. Parmis les aventures, je suis passé par une SPA (Single Page Application) mais contrairement à SvelteKit, ce n'est malheureusement pas géré nativement par Astro. C'est pourquoi il a fallu passer par un plugin pour Astro : astro-spa. Cependant, après de nombreuses batailles pour correctement exécuter mes scripts et plus de code consacré à gérer les problèmes de la spa que de code en soi, j'ai pris la décision d'enlever la spa malgré les avantages en termes de performance qu'elle apporte.

Il reste un petit détail maintenant, pour rendre le site un peu plus vivant, j'ai décidé d'utiliser Lenis, une librairie pour rajouter des animations avec le scroll. Malgré quelques connaissances avec la librairie locomotive scroll, son manque d'accessibilité et ses besoins gourmands en performance m'ont fait essayer cette nouvelle librairie. Lenis est la base de la page « blog »mais la librairie étant pauvre, toute la logique a dû être implémentée.

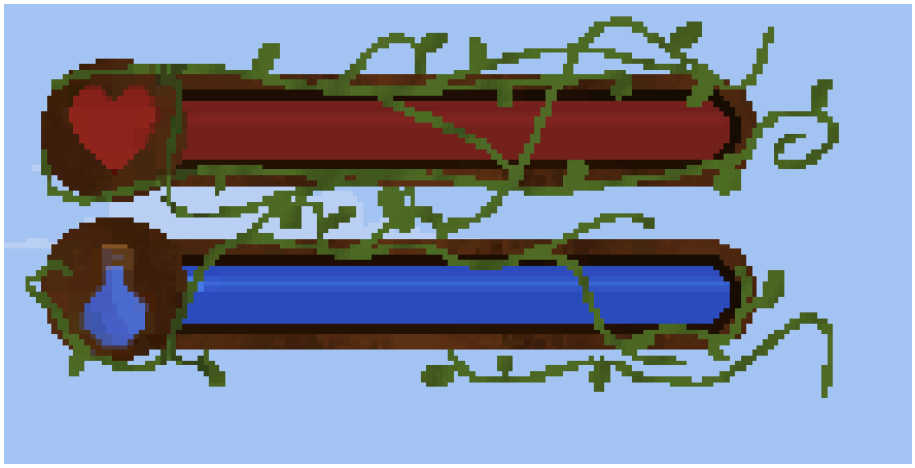

FIGURE 68 – Barres de vie et de mana

FIGURE 69 – Barre de sorts

7.1.2 Le style
Antoine, Soutenance 1

Des essais précédents, la leçon a été retenue : Ne pas en faire de trop. Cependant, même si le site se veut sobre, j'ai incorporé quelques touches qui rappellent le jeu. Par exemple, pour mettre l'utilisateur dans l'ambiance d'un jeu vidéo, l'utilisation d'une police en pixel art est pertinente, précisément quand le jeu lui-même est en pixel art. Pour rester dans cet esprit du pixel art, beaucoup d'éléments rectangulaires sont implémentés en évitant d'arrondir les coins des objets. De cette manière, tout est un petit peu « carré », à la manière du pixel art. En plus de rappeler la forme du jeu, je voulais aussi en rappeler l'esprit. Pour évoquer la magie présente dans Hermita j'ai pensé à une fumée bleue qui suit la souris de l'utilisateur. (visible dans la figure ci-dessous)

Et finalement, avec la forme et l'esprit du jeu présents dans le site, il ne manque plus qu'à montrer un bout du monde d'Hermita. Pour cela, le monde étant peuplé de golems, le tout dans un style légèrement steampunk, j'ai pensé à mettre des roues dentées en bas à droite de l'écran ; qui tournent avec le scroll. En plus de cela, je compte implémenter des îles flottantes tirées de notre jeu qui bougeraient avec un effet de parallaxe, comme sur la page d'accueil (qui a des exemples avant de recevoir des images plus finies du rendu du jeu).

7.1.3 La page de blog
Antoine, Soutenance 1

Le blog prend la forme d'une frise chronologique avec un scroll horizontal, courant dans les sites modernes qui se veulent « classe ». J'ai dû beaucoup bidouiller avec Lenis en écrivant une fonction qui devinait quelle poste était visible à l'écran à tel endroit du scroll. Ceci est

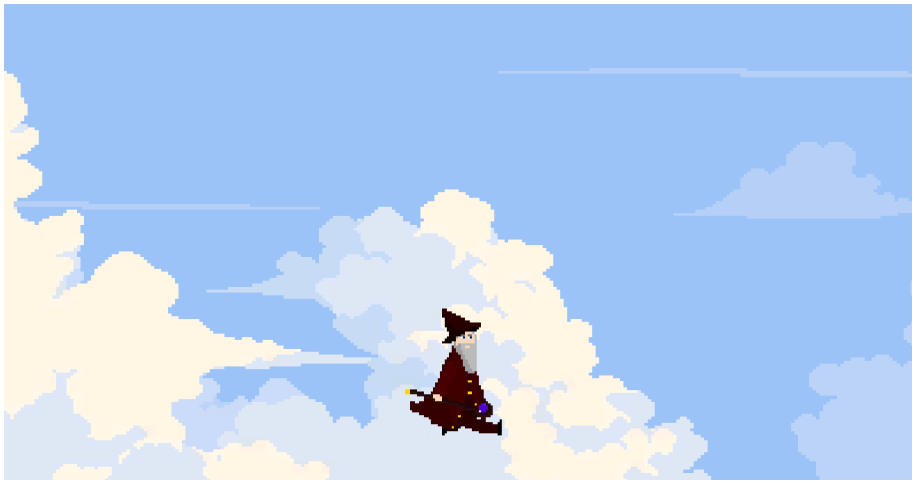

FIGURE 70 – Écran de chargement

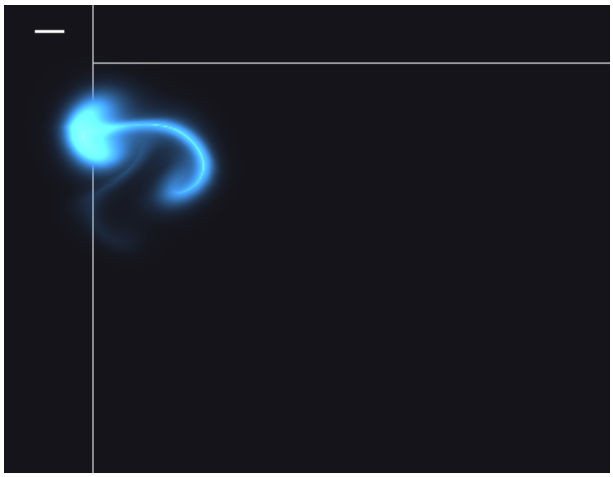

FIGURE 71 – Effet de fumée

nécessaire pour faire la petite animation et pour mettre la date et l'image de chaque poste dans la zone d'affichage commune à tous les posts. Entre les posts se trouvent des carrés de petite taille, ils représentent le nombre de jours passés entre chaque post et une élipse est représentée lorsque le nombre de jours dépasse 10. On peut ensuite cliquer sur chaque étiquette de post pour lire le texte. Cette ouverture se fait avec une transition poussée pour donner un effet plus vivant.

7.1.4 Page d'accueil
Antoine, Soutenance 2

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 73, la section est composée d'un simple texte avec des dalles (voir 4) 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

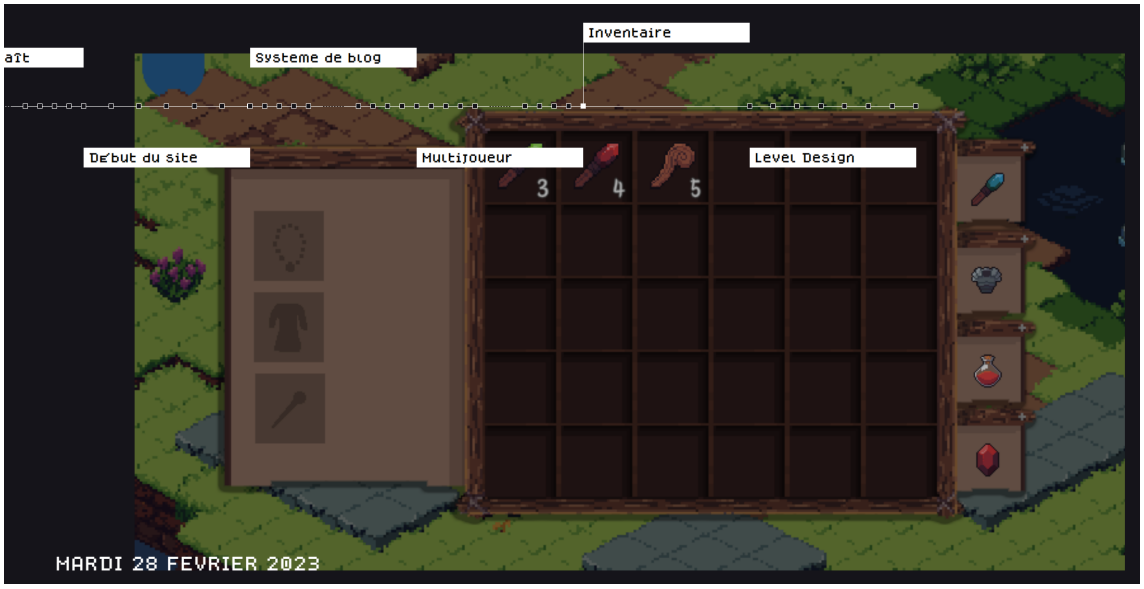

FIGURE 72 – Frise chronologique

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.

7.1.5 Page de téléchargement
Antoine, Soutenance 2

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

7.1.6 Performaces
Antoine, Soutenance 2

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 75. 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 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*.

FIGURE 73 – Page d'accueil

7.1.7 Page de crédits
Antoine, Soutenance 3

On ne veut pas que Hermita seulement soit mondialement connu, son équipe doit aussi l'être. Il est donc important de mettre en avant les personnes qui ont travaillé sur le projet. Je pense bien sûr aux membre d'Epistars mais aussi à toutes les petites mains extérieures à l'EPITA qui ont contribué au projet que ce soit par la réalisation de sprites, de musiques ou par leur travail de Beta-testeur. Chaque membre possède une petite description et une image de lui. Ces éléments prennent la forme de cartes qui peuvent être retournées pour afficher le texte. Pendant l'animation de retournement, un effet de profondeur sur les éléments est révélé.

7.2 Instagram

Dylan, Soutenance 3 Il fallait bien trouver une solution pour que Hermita devienne réellement populaire. C'est ainsi qu'est venu l'idée de créer un compte Instagram pour notre jeu ! Celui-ci n'a peut-être pas atteint le nombre d'abonnés qu'il mérite (33 au moment où j'écris ceci), mais il est très apprécié de nos amis, qui suivent l'avancé du projet ! Il n'est bien sûr pas trop tard pour s'abonner, alors foncez ! Le nom du compte est à retrouver sur la figure 76.

7.3 Discord

Baptiste, Soutenance 1 Dès le début du projet, beaucoup de nos proches ont été motivés à nous aider à la création de celui-ci, nous avons donc décidé d'utiliser discord pour les regrouper, cet outil nous a principalement été utile lors des sessions de tests car cela a permis de trouver énormément de bugs. Celui-ci nous servait à communiquer avec toutes les personnes qui nous ont aidé pour ce projet mais aussi au sein du groupe dans des salons privés.

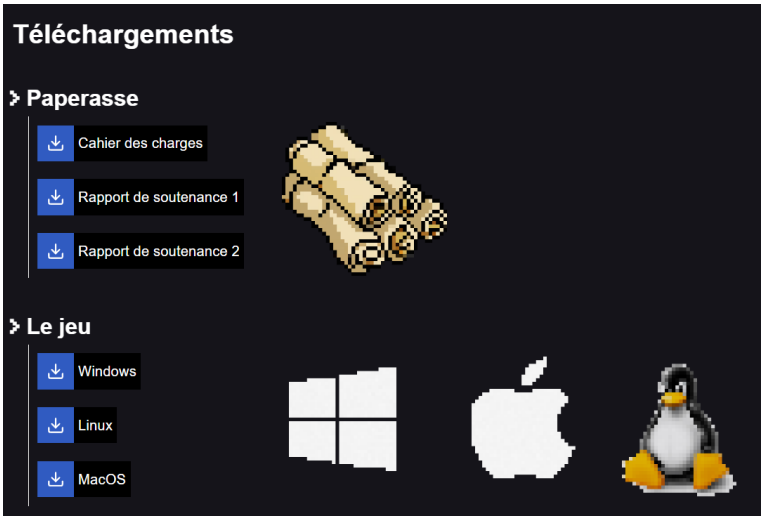

FIGURE 74 – Page de téléchargement

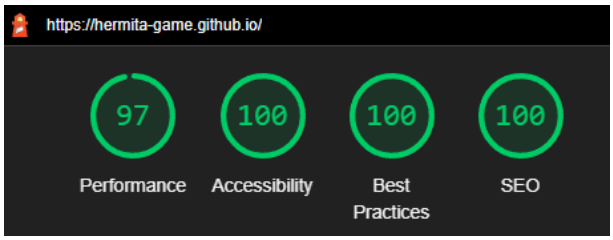

FIGURE 75 – Performances du site

7.4 Autour du jeu

7.4.1 Installateur
Antoine, Soutenance 3

Il faut bien sûr un installateur pour le jeu. La solution la plus simple est d'utiliser *Inno Setup* qui est un logiciel gratuit et très complet. Il permet de créer des installateurs avec des options de personnalisation assez poussées. Il inclut aussi un désinstalleur qui permet de supprimer le jeu proprement tout en gardant les sauvegardes.

8 Conclusion

Le jeu Hermita fut un projet ambitieux, qui avait pour but d'offrir une belle expérience aux joueurs. Notre vision était avant tout d'offrir un jeu simple d'utilisation, accessible à tous, tout en offrant un design élaboré, des possibilités de jeu infinies, et une histoire époustouflante.

L'ensemble du cahier des charges a été respecté, que ce soit le multijoueur, l'IA, le système de craft et le système de combat, les designs, la carte, et bien d'autres ! Nous avons même réussi à aller plus loin, en implémentant des musiques par exemple, qui rendent notre jeu unique ! Si nous sommes allés plus loin dans certains détails, c'est avant tout parce que ce jeu n'était plus qu'un simple projet, mais un projet passionnant !

Toutes les personnes qui ont testées notre jeu (notamment nos bêta-testeurs) sont satisfaits

FIGURE 76 – Notre page Instagram

de leur expérience. Ils nous ont aussi énormément aidé pour déceler le moindre petit bug que pouvait contenir notre jeu durant sa production.

Pour notre part, malgré les différents problèmes de communications au sein de notre équipe avec un des membres, nous sommes très satisfaits de ce que nous avons créés. Cela a été pour nous une expérience très enrichissante, nous avons appris de nouvelles compétences, découvert de nouvelles passions, et amélioré certaines de nos connaissances, notamment notre créativité. Cette expérience restera pour nous inoubliable, et il n'est pas improbable que nous continuons par la suite le développement de ce jeu en allant au-delà des demandes du cahier des charges.

Le jeu est téléchargeable dès maintenant sur le site.