

Rapport de projet

OUMEZIANE Mustapha, LOUP Thomas, RABARISON Tiana Mélanie

Mai 2023

REVENGE OF AKIRA



P l a n :

1. Reprise du cahier des charges.....	3
1.1. Introduction.....	3
1.2. Présentation du groupe.....	3
1.3. Explication des tâches.....	3
1.4. Répartition des tâches et avancement.....	5
2. Présentations des tâches réalisés (chronologique).....	7
2.1. Réflexion sur le Gameplay (tous).....	7
2.2. Implémentation de la physique et animations(Thomas).....	8
2.3. Création de la map(Mustapha).....	12
2.4. Implémentation de l'inventaire(Tiana Mélanie).....	15
2.5. Implémentation des Items (Tiana Mélanie).....	17
2.6. Implémentation des interfaces(Thomas).....	18
2.7. Implémentation du son (Mustapha).....	21
2.8. Finalisation du menu (Thomas).....	22
2.9. Création de la deuxième map (Mustapha).....	24
2.10. Mise en place de la sauvegarde (Thomas).....	30
2.11. Implémentation des premières intelligences artificielles (Tiana Mélanie et Thomas)..	33
2.12. Améliorations de l'inventaire et des items (Tiana Mélanie).....	35
2.13. Implémentation de différent menu(Thomas).....	38
2.14. Implémentation du multijoueur (Thomas).....	39
2.14.1. Première version.....	40
2.14.2. Deuxième version.....	41
3. Récit de la réalisation.....	44
4. Industrialisation.....	45
4.1. Présentation du Site Web.....	45
4.2. Plan du Site Web.....	45
4.3. L'installateur.....	46
5. Synthèses personnelles.....	48
6. Conclusion.....	49
7. Logiciels et outils utilisés.....	50
8. Annexes.....	51

1. Reprise du cahier des charges

1.1. Introduction

Pour le projet du deuxième semestre, nous nous sommes réunis sous le nom de *Calypso pour proposer un A-RPG immersif dans un univers médiéval. Le principe d'un* A-RPG est globalement celui d'un RPG : le joueur incarne généralement un seul personnage et le jeu se passe en temps réel. La majorité de ces jeux se déroulent à la troisième personne, avec une caméra plus proche du personnage que dans les jeux de rôle traditionnels. Le joueur évolue librement dans un monde et doit réussir certains objectifs pour avancer dans l'aventure. De plus, la plupart des A-RPG proposent un système d'évolution du personnage poussé (équipement, caractéristiques, niveau d'expérience, etc.).

La principale différence avec les jeux de rôle classiques est que les A-RPG se veulent plus dynamiques. Tous les déplacements se placent sur un même plan et les ennemis sont directement visibles. Les combats s'engagent dès lors que l'ennemi est à portée et se déroulent dans le même environnement (contrairement aux RPG japonais classiques où les phases d'exploration et de combat ont lieu dans des zones distinctes).

Notre projet consiste en la création d'un jeu sur Windows nommé *Vengeance of Akira*. Le jeu sera alors jouable avec le clavier de l'ordinateur, seul ou avec une autre personne possédant une autre machine connectée au même réseau.

Nous nous sommes rapidement mis d'accord sur des points comme le style graphique, le son ou encore le type d'univers (dans notre cas médiéval-fantastique) et le scénario du jeu pour parler de l'histoire du héros dont le nom est Akira. Par ailleurs, ce jeu vidéo consiste à l'exploration d'un jeu immersif, et il sera codé en Csharp et réalisé avec l'outil Unity.

Enfin, nous avons établi un planning d'avancement des tâches pour faire avancer au mieux notre projet.

1.2. Présentation du groupe

Le groupe est aujourd'hui constitué de 3 personnes suivie du départ d'un membre du groupe avant la première soutenance :

- OUMEZIANE Mustapha (chef de projet)
- LOUP Thomas
- RABARISON Tiana Mélanie

1.3. Explication des tâches

Décors et environnement:

Le jeu est divisé en 3 zones:

- La première zone sera un village en guise de tutoriel.
- La deuxième sera constituée d'une forêt, plus difficile à franchir, principalement composée d'arbres et de collines vertes avec des monstres. On aura ensuite le château qui est une zone difficile et labyrinthique qui mène à la zone finale on peut y trouver des monstres mais aussi des items. Et enfin une zone secrète avec un boss optionnel.
- La troisième zone est la partie multijoueur

Personnages du jeu:

Liste des entités du jeu:

- villageois
- le personnage principal (le joueur)
- des monstres
- le roi (boss)
- le bosse optionnel

Physique:

Le joueur incarne un personnage dans un environnement en trois dimensions. Il doit pouvoir y naviguer facilement, c'est pourquoi les fonctionnalités suivantes doivent être implémentées:

- déplacements multidirectionnels (avant / arrière, gauche / droite)
- rotation de la caméra pour pouvoir regarder partout autour de soi
- fonction pour courir, sauter et esquiver
- la gravité doit être prise en compte
- la collision avec les obstacles doit être prise en compte

Réseau:

Pour la partie réseau, on va proposer au joueur d'accéder à une arène pour combattre d'autres joueurs, cela permettra aussi de faire des combat entre des amis pour savoir qui est la plus à la plus d'expérience sur le jeu.

Intelligence Artificielle:

Au cours du jeu le joueur va pouvoir affronter des ennemis et il pourra croiser différents villageois. Ils seront implantés par une intelligence artificielle. Les ennemis attaqueront lorsque le joueur se trouvera dans une certaine zone autour que l'on définit et sinon ils effectueront divers actions. Les villageois joueront des animations de façon aléatoire.

Le son:

Le son se caractérise en trois catégories principales :

- L'ambiance sonore est composée d'effets tels que les bruits de pas, les sons d'extérieur (feuillage, bruits de pas) donnant plus de réalisme et de crédibilité au jeu.
- Les effets sonores d'interfaces utilisés dans les différents menus ou comme indicateurs lors de moments clés de la partie.
- On intégrerait des musiques de bosses lors des combats, mais aussi des musiques de fond le long du jeu.

Mécanique du jeu:

Les mécaniques de jeu correspondent aux fonctionnalités du jeu vidéo. C'est une partie primordiale du projet qui vise à coder les règles et définir les mécanismes d'interaction entre le joueur et les objets.

Cela consiste à faire de la Programmation Orientée Objet en Csharp pour implémenter les différentes armes et items, l'accès au nouvelle zone de la carte. En effet, il sera nécessaire d'avoir des classes pour chaque objet, hérité de classe parent. On aura aussi un inventaire avec différents objets.

Enfin, toute la partie combat et monstre fait partie de la mécanique du jeu. Les monstres seront aussi implémentés par de la Programmation Orientée Objet.

Expérience utilisateur:

L'expérience utilisateur représente l'interface liant le jeu au joueur, et la flexibilité du programme quant à sa personnalisation, par le biais de paramètres par exemple.

On y retrouve les différents menus permettant de naviguer dans le jeu de façon simple, de pouvoir sauvegarder une partie, mais aussi l'attribution des contrôles et actions du jeu (choisir les touches de déplacement du personnage, touches d'action et volume sonore).

L'expérience utilisateur est la première interaction du joueur avec le programme : elle doit donc être compréhensible et facile d'utilisation pour ne pas déstabiliser le joueur.

Sites Web:

La création d'un site internet sera là pour présenter le projet. Ce site indiquera les différentes étapes liées à la création du jeu, une démonstration du jeu, une présentation de l'équipe et un accès direct pour le télécharger. La conception du site demandera des compétences dans des langages web tels que le HTML et le CSS avec un peu de JavaScript si des animations sont requises.

L'hébergement se fera sur un service gratuit comme Firebase ou Github, le coût d'un nom de domaine est à prévoir.

1.4. Répartition des tâches et avancement

	Mustapha	Thomas	Mélanie
décors et environnement	Responsable		suppléant
Modélisation des personnages	Responsable		suppléant
Physique	suppléant	Responsable	
Réseau		suppléant	Responsable
Intelligence artificielle	suppléant	Responsable	
Son	Responsable		suppléant
Mécanique du jeu		suppléant	Responsable
Expérience utilisateur		Responsable	suppléant
Site WEB		suppléant	Responsable

La répartition des tâches a été globalement bien tenue, quelque tache on un peu changé, au niveau du réseau c'est Thomas qui en a été le responsable avec Mélanie. Et pour l'intelligence artificielle c'est Mélanie qui est devenue la suppléante.

Les différentes tâches ont été faites avec rigueur et sérieux par les membres du groupe, ce qui a permis un bon avancement comme celui défini dans le cahier des charges. Quelques retards sont survenus principalement pour la deuxième soutenance du au départ d'un membre du groupe. Le membre est parti avant la première soutenance mais cela nous a jouer pour la deuxième soutenance car pas mal de chose était prévu est certaine ont été sous estimé comme la sauvegarde.

Mais nous avons quand même fini de réaliser les différentes tâches que nous avions prévu pour ce jeu.

2. Présentations des tâches réalisé (chronologique)

2.1. Réflexion sur le Gameplay (tous)

Pour commencer à vouloir produire quelque chose, on a d'abord réfléchi au gameplay du jeu que l'on voulait implémenter. Pour cela on a posé les différentes idées trouvées puis nous avons fait un choix sur ce que nous allions faire.

Lorsqu'on ouvre le jeu on a voulu avoir un menu interactif avec la possibilité au joueur de personnaliser son gameplay. C'est à travers ce menu que l'on pourra lancer une nouvelle partie ou charger une partie déjà sauvegardée.

Une fois la partie lancée, on débute dans un village assez simple et facile à finir. Le but est principalement de faire découvrir les commandes, les mécaniques de jeu et le style du jeu. On y trouve quelques items comme des potions qui restaurent des pv et qui augmentent les dégâts sur une courte durée mais elles sont cachées ! Tout comme les armes disponibles dans ce premier niveau. Le joueur apprend donc à se déplacer, à explorer mais aussi à combattre car après avoir battu le monstre de cette zone l'utilisateur sera récompensé d'une arme plus puissante et pourra accéder à la suite du jeu. A partir de ce moment-là, il pourra aussi accéder au multijoueur et combattre dans une arène contre d'autres personnes.

Tout au long du jeu, le joueur pourra augmenter en niveau en tuant des monstres ou en faisant des quêtes, cela lui permettra d'augmenter ses attributs comme la force qui lui procurera plus de dégâts ou même l'augmentation de ces points de vie. Il aura également accès à un menu lorsqu'il joue, en appelant ce menu il mettra pause au jeu et pourra voir les réglages, revenir au menu principal, sauvegarder la partie et accéder au multijoueur.

La deuxième zone est une forêt avec un niveau de difficulté intermédiaire avec des monstres de forme bestiale comme des trolls. La difficulté de cette zone réside surtout dans le sens de l'orientation car la forêt est dense et qu'il y des portes de sortie différentes.

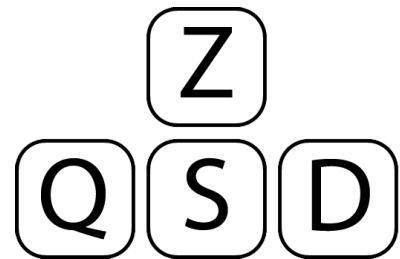
La première sortie de zone mène au châteaux, qui est la zone la plus difficile du jeu où se trouve le Boss finale, cette zone regorge de monstres dans tous les recoins mais il y également des items surpuissants pour pouvoir arriver à bout de la zone cependant ces items se mérite mais il est tout à fait possible de finir le jeu avec n'importe quelle arme. Le Boss finale se trouve au niveau de la salle du trône et qui est au passage la cible principal de Akira. Une fois le Boss final vaincu, le joueur aura fini le jeu. Mais ça ne s'arrête pas là car le joueur peut revenir sur ses pas pour continuer son exploration ou combattre d'autres joueurs en PVP via le mode multijoueur. Précédemment nous avons dit qu'il y avait plus sorti dans la forêt, la deuxième sortie et plus discrète donc plus difficile à trouver mais elle nous conduit dans un endroit du château inaccessible depuis la voie principale, tandis que la dernière sortie est la plus difficile à trouver et elle nous mène vers une zone caché appelé "Ruines oubliées" ou se cache un boss plus dur que le boss final et son butin est l'arme la plus puissante du jeu. Cette zone a été pensée pour les joueurs ayant soif de challenge..

2.2. Implémentation de la physique et animations(Thomas)

Les fonctionnalités essentielles au déplacement d'un joueur sont de pouvoir avancer, reculer mais aussi tourner à gauche et à droite et enfin sauter. Nous avons également l'action d'attaquer comme fonction capital, chacune de ses actions sera accompagnée d'animations. Pour la première soutenance, ces fonctionnalités ont été implémentées: le joueur peut donc désormais se déplacer et regarder autour de lui, attaquer et courir.

Fonctionnement du déplacement:

Les déplacements du personnage se feront avec les touches directionnelles du clavier ZQSD. Ces quatre touches vont fonctionner par paires de deux, Z (respectivement S) pour avancer (respectivement reculer), et Q (respectivement D) pour tourner à gauche (respectivement à droite). Chaque paires renverra une valeur Arbitraire positive, négative ou nulle.



Par exemple, lorsqu'on va appuyer sur la touche Z on renvoie 1, cela signifie que sur l'axe avant-arrière la position sera modifiée vers le côté positif de l'axe (le joueur avance). Quand on appuie sur S on renvoie -1 donc la position change vers le côté négatif (le joueur recule). On retrouvera le même système pour les touches Q et D avec respectivement -1 (le joueur tourne vers la gauche) et 1 (le joueur tourne vers la droite). Grâce à ce système, si on peut appuyer sur les deux touches d'une même paire en même temps, les valeurs seront alors annulées. On peut aussi grâce à cette implémentation appuyer sur deux paires en même temps, c'est-à-dire avancer et tourner en même temps.

Pour mettre en place ces mouvements, on va récupérer les valeurs arbitraires pour en faire des "vector3", c'est ce que l'on peut appeler une force, c'est un vecteur avec 3 coordonnées et elle permettra au joueur de se déplacer. Ce vecteur a donc trois directions, qui sont ses axes avant-arrière (ainsi que gauche-droite et haut-bas, d'où le nombre 3) sur lequel le joueur se déplace. Le sens du vecteur est calculé grâce aux valeurs arbitraires citées précédemment.

Si on reprend l'exemple plus haut, en appuyant uniquement sur Z (on renvoie 1), un vecteur est créé sur l'axe avant-arrière avec comme valeur, la valeur arbitraire. Enfin il restera à appliquer le vecteur au personnage pour le pousser vers l'avant.

```
Vector3 mouvement = new Vector3(x: 0f, y: 0f, z: move);
```

Vector3 pour la touche Z et S, move correspond à la valeur arbitraire et il est dans z qui correspond a l'axe avant-arrière

Ensuite, pour le pousser on effectue une translation du personnage avec sa vitesse et le vector3 trouvé. On obtient donc :

```
gameObject.transform.Translate(translation: mouvement * (walkSpeed * Time.deltaTime));
```

gameObject correspond au personnage

Dans le calcul on trouve Time.deltaTime, cette valeur permet par exemple si on voulait faire un ralentie sur une action de ralentir automatiquement le mouvement si le temps est modifié (de la même manière on peut accélérer le temps).

La méthode pour faire courir le joueur utilisera le même système, le seul changement sera au niveau de la vitesse de déplacement ainsi que sur les touches appuyer. En effet pour courir le joueur devra appuyer sur la touche Z et Shift en même temps.

Le joueur était initialement équipé d'un "RigidBody". C'est une structure fournie par Unity permettant de contraindre le joueur aux forces physiques les plus communes: on peut lui attribuer une masse, lui appliquer une force de gravité. Nous l'avons utilisé pour cela car il permet de créer lui-même une force vers la bas pour simuler la gravité. C'est avec cette structure que nous avons implémenté le saut. Définie par la touche espace du clavier, on va pousser le personnage sur l'axe Haut-bas avec un vecteur composé d'une hauteur de saut. Pour ne pas sauter à l'infinie il faut d'abord regarder si on touche le sol avant de sauter et si on est assez proche du sol on peut effectuer le saut sinon la gravité du "RigidBody" fera effet.

On a aussi ajouté au joueur un composant la "Capsule Collider", elle génère une capsule autour du personnage et permet au joueur de ne pas avoir de collision avec les obstacles (maison, arbre, monstre ...). C'est aussi cette structure qui permet de ne pas traverser le sol.

Fonctionnement des animations:

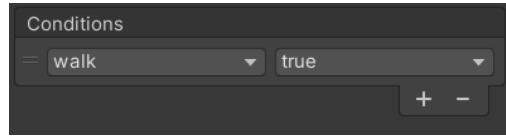
Pour donner plus de dynamique au gameplay du jeu nous avons ajouté différentes animations en fonction de l'action effectuée. Pour cela nous avons principalement utilisé le site web "<https://www.mixamo.com>" qui propose des models 3D mais aussi un large choix d'animations pour le déplacement mais aussi pour d'autres actions d'interaction (attaquer, animation de mort ...).

Pour les gérer, Unity propose plusieurs méthodes. La première méthode que nous avons utilisée en premier était de jouer les animations dès qu'il le fallait, c'est-à-dire que l'on lance l'animation dans le code source correspondant (quand on appuie sur Z on lance l'animation de marche ...).

```
_animation.Play(animation: "walk");
```

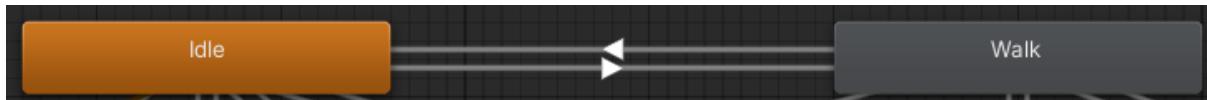
On lance l'animation "walk" qui permet de marcher

Bien que cette méthode soit simple à utiliser, on y trouve un point négatif important, c'est l'absence de transition entre les animations. En effet, lorsqu'un joueur verra son personnage s'arrêter brusquement, changer d'animation d'un coup cela n'est pas très agréable à regarder. Pour arranger cela, la deuxième méthode utilise l'Animator, c'est une structure d'Unity qui permet de lier les animations entre elles par des transitions. Pour jouer une animation il va falloir déclencher une transition, pour cela on peut y mettre des conditions. Nous avons choisi d'utiliser des booléens pour ces conditions car c'est ce qui nous paraissait le plus intuitif pour commander des actions.



Emplacement des conditions, ici pour la transition qui amène à l'animation de marche

C'est cette nouvelle méthode que nous avons utilisé ensuite, et que nous avons gardé jusqu'à la fin et amélioré par la suite.



2 transitions entre l'animation "idle" et "walk", une pour aller de "idle" vers "walk" et l'autre pour le chemin inverse

Explication d'un cas:

Au départ, tous les booléens étaient initialement faux.

Le joueur veut avancer il appuie sur Z, pour lancer l'animation de marche il faut donc activer la transition qui correspond (notons-la "walk", marche en anglais), on met "walk" à vrai puis l'animation se lance et quand on appuie plus, il se remet à faux pour arrêter l'animation.

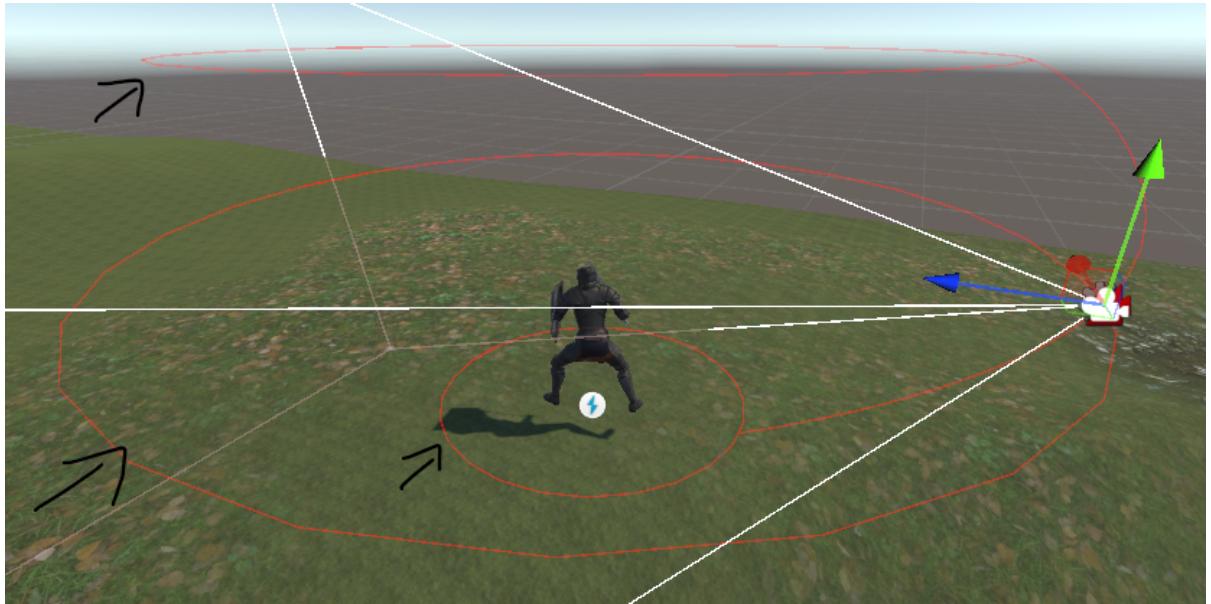
```
_animator.SetBool( name: "walk", value: true);
```

mise à vrai du booléen "walk"

Ce système d'animation nous offre la possibilité de choisir si on veut qu'une animation soit jouée entièrement ou si on peut l'interrompre. Dans le cas précédent, on veut interrompre la marche dès que l'on n'appuie plus sur Z, contrairement au saut ou à l'attaque qui doivent se jouer en entier.

Un point contraignant sur notre façon d'utiliser l'Animator c'est la compréhension, en effet plus on va ajouter d'animation et donc de transition, plus ce sera compliqué de s'y retrouver. Mais comme nous n'avons pas utilisé un nombre important d'animation (7 animations pour le personnage), nous n'avons pas rencontré de difficulté pour le comprendre.

Le joueur pourra également contrôler la caméra avec la souris, ce qui permettra de diversifier encore plus le gameplay du jeu. Pour cela on a utilisé une structure d'Unity la "Cinemachine", elle va permettre de pouvoir bouger la caméra autour d'une cible sélectionnée en fonction du pointeur de la souris. Grâce à cela on va pouvoir tourner autour du personnage librement sans déplacer le joueur ce qui correspond bien à nos attentes. La "Cinemachine" nous permet donc d'identifier un objet (ici le personnage) à regarder, à suivre et de définir sans script où la caméra peut se déplacer autour du joueur.



Affichage quand on sélectionne la “Cinemachine”

On peut voir sur cette image trois cercles avec des liaisons les unissant. Ces cercles représentent les trois principaux stades où la caméra peut se déplacer. Nous avons choisi des cercles avec des diamètres assez importants au milieu pour permettre au joueur d'avoir une bonne vue sur ce qu'il se passe dans les alentours. Cela permet aussi de ne pas avoir des problèmes comme voir sous le sol, voir dans le personnage... Cette caméra a aussi la possibilité d'avoir un “collider” pour ne pas traverser les objets. Par exemple, si notre joueur est dans une forêt on ne veut pas que la caméra traverse les arbres, on a donc mis en place un “collider” et on l'a paramétré en sorte que si la caméra à un obstacle elle se rapproche du personnage avec une limite définie.

Actuellement, à la fin de la conception du jeu nous utilisons toujours cette implémentation car elle est très utile, pas très compliqué à mettre en place et ne comporte que très peu de script. C'est pourquoi nous l'avons choisie et gardée tout au long du projet.

2.3. Création de la map(Mustapha)

Pour créer la map, il faut dans un premier temps délimiter la zone du premier niveau. Nous sommes partis sur un terrain rectangulaire et allongé de sorte à orienter le joueur vers le bout de la zone. Tous les game objects ont été importé de sites fournisseur et loble de droit tel que unity asset store et mixamo.

Terrain Width	150
Terrain Length	250

Ensuite, on rajoute les reliefs pour donner un aspect de plaines, des arbres, les maisons et les textures pour créer un environnement réaliste.

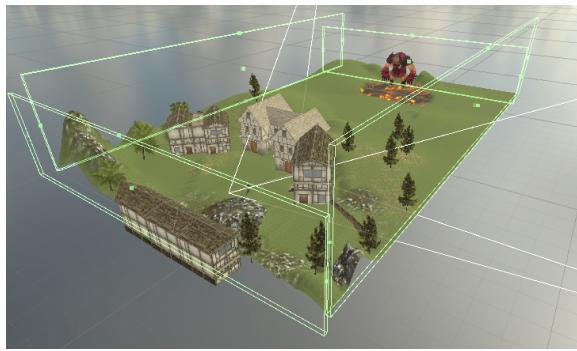


Pour apporter encore plus de réalisme, nous avons importé un SkyBox qui, comme son nom l'indique, nous emboite dans le ciel.

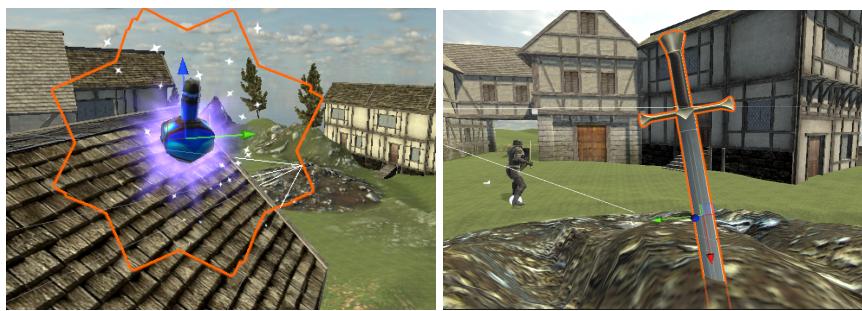


Une des chose importante à ne pas négliger dans la création du décor à était les limites à ne pas franchir car sinon le joueur sortirait de la map et tomberait dans le vide. Pour y remédier, nous avons seulement implémenté des murs avec un box collider pour bloquer le joueur et

nous avons retiré le mesh renderer ce qui permet de rendre le mur invisible.

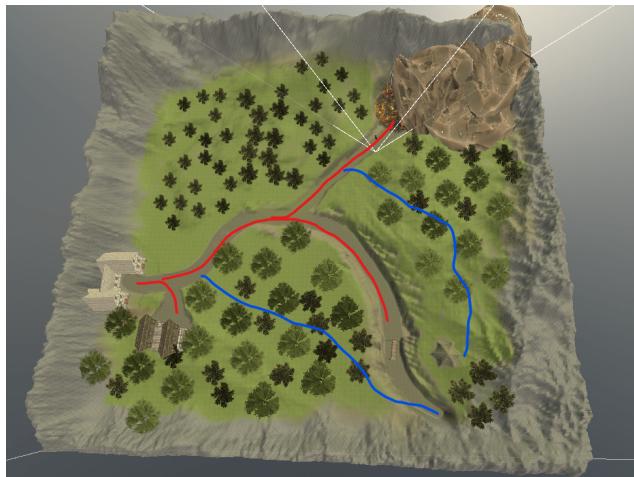


Pour finir, nous avons disposé des items cachés un peu partout dans le map, pour récompenser le joueur de sa curiosité à explorer les moindres recoins.



Pour la finalisation de notre jeu, nous avons pris la décision de refaire entièrement les deux map car elles ne convenaient pas à nos attentes. les objectifs restent les mêmes, une map de tutoriel avec un mini boss à la fin, puis la map finale avec le château, le boss optionnel et les objets cachés.

Commençons par la map tutoriel:



map tutoriel version 2 vu du dessus vision à hauteur d'homme

Les chemins en rouge sont les voies principales pour finir le niveau mais si le joueur veut aller un peu plus loin il est d'explorer la zone et de découvrir de nouveaux items qui rendent le joueur plus puissant ces chemins sont plus difficile d'accès et sont représentés en rouge il y a une créature à tuer dans la grotte en haut à droite ce qui nous permettra de passer au niveau suivant. Certains indices aident le joueur et le poussent à explorer davantage les recoins comme par exemple ces fleurs rouges qui forment un chemin pour attirer l'attention du joueur et qui vont le mener à un item caché.



feurs rouge qui guide le joueur —————-> arrivé de l'énigme

2.4. Implémentation de l'inventaire(Tiana Mélanie)

Grâce à l'implémentation de la physique, l'implémentation de l'inventaire a pu se mettre en place.

L'inventeur du jeu se fait par la création d'un nouveau Canva et ensuite d'un scroll View où l'on insère des contents, qui sont des boutons . Ce qui a pu permettre :



Pour l'inventaire, on a modélisé deux boutons principales notamment :

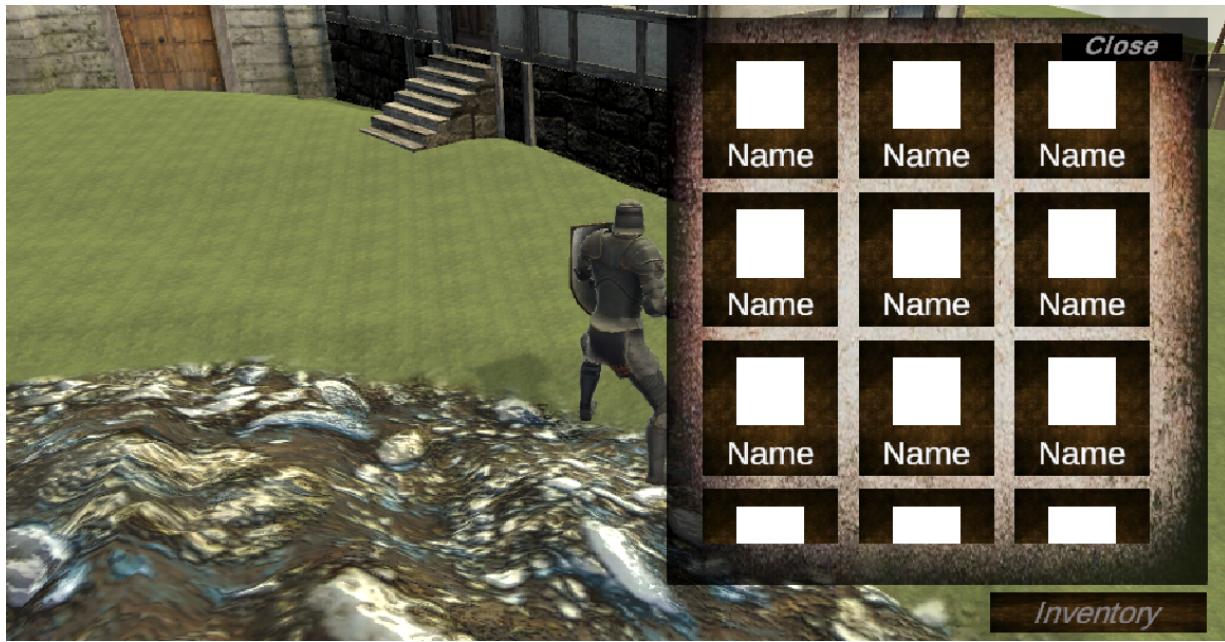
-inventor = permet d'accéder à l'inventaire du joueur

-close = qui permet de fermer l'inventaire

Ce qui nous permet d'avoir cette interface dans game :



et celle-ci lorsque le bouton "Inventory" est appuyé :



Dans L'inventaire, on retrouve des petits cubes avec un bouton, et un nom. C'est les places respectives de chaque élément dans l'inventaire que le joueur va récupérer au fil du temps
Dans la partie implémentation de l'inventaire le joueur peut prendre et mettre dans l'inventaire des items.

Pour l'implémentation de l'inventaire, le principe étant de permettre au joueur de ramasser un objet et de le stocker dans son inventaire.

On a donc choisi d'utiliser la touche "E" du clavier pour permettre au joueur de prendre l'objet.
Mais on a rencontré des difficultés qu'on doit essayer de régler pour la soutenance prochaine.
Pour ça on a dit d'utiliser Raycast hit.

On a codé le script ci dessous :

```
if (Input.GetKeyDown(KeyCode.E))
{
    inventoryManager.content.Add(hit.transform.gameObject.GetComponent<ItemController>().item);
    Destroy(hit.transform.gameObject);
}
```

La fonction Pickup permet ainsi de mettre dans l'inventaire les items qu'on décide de prendre par la touche "E", et de la supprimer visuellement sur la Map.

2.5. Implémentation des Items (Tiana Mélanie)

Comme chaque RPG, on a dû implémenter des items. On a implémenté deux types d'Items: les potions et les épées.

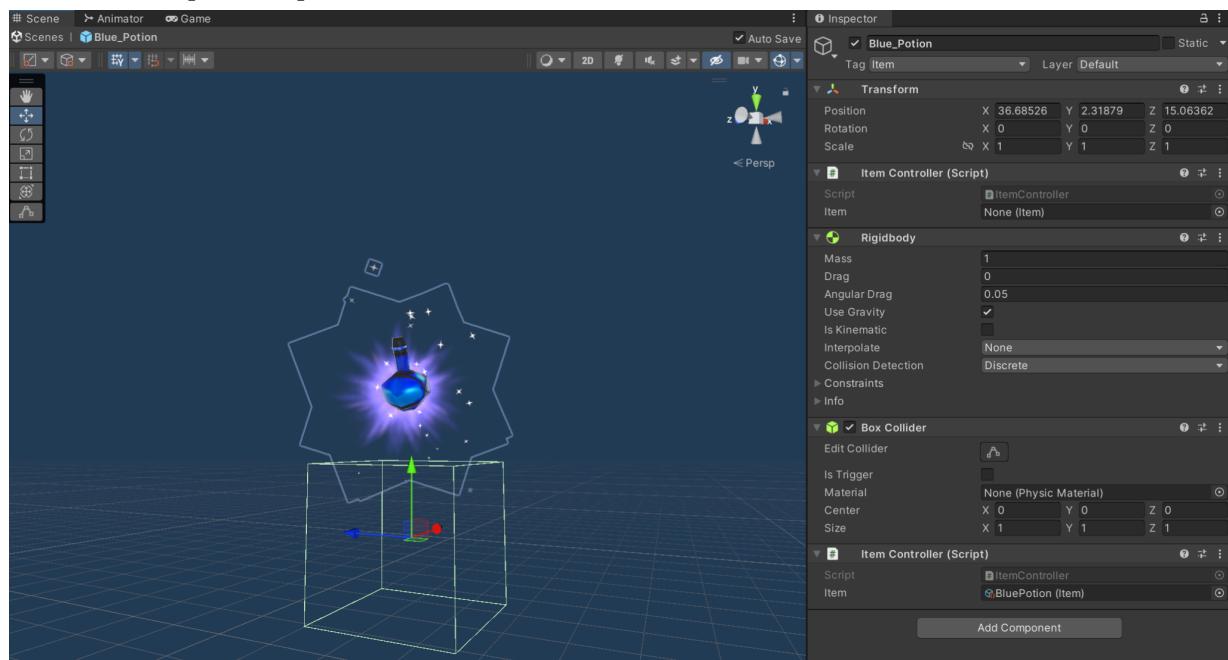
Ainsi chaque item a pour caractéristique son nom, sa valeur, son icon , son type ainsi que sa prefab.

```
[CreateAssetMenu(fileName = "Item", menuName = "Items/create new Item")]
public class Item : ScriptableObject
{
    public int id;
    public string itemName;
    public float value;
    public Sprite icon; // icon de l'item
    public ItemType itemType;
    public GameObject prefab;

    public enum ItemType //définir les Type de l'item
    {
        Potion,
        Sword
    }
}
```

Ainsi pour permettre à au joueur de prendre les potions, on a du ajouter un Box collider et un **Rigidbody** pour que le personnage reconnaît que c'est un item.

Voilà un exemple d'inspector d'un item:



2.6. Implémentation des interfaces(Thomas)

Les interfaces utilisateurs sont des éléments importants du jeu. Elles permettent une personnalisation du jeu, et guident le joueur sur sa prise en main et son gameplay. Pour ce faire, des menus simples permettant de réaliser des interactions basiques ont été implémentés pour la première soutenance et ont été développés par la suite.

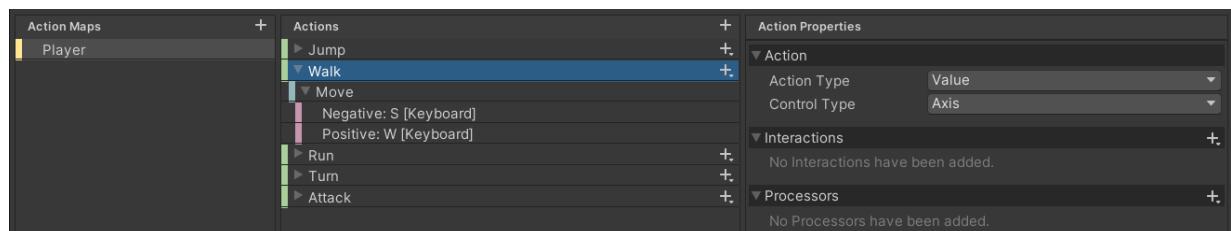
Au lancement du jeu, un menu principal s'affiche dans lequel le joueur peut effectuer diverses actions. Il peut lancer une partie, accéder aux options pour les modifier comme il le veut (les différentes options sont expliquées par la suite). Il y a également un bouton pour quitter le jeu.

Cette première version du menu ne prenait pas en compte de pouvoir charger une partie déjà commencée, ceci a été fait pour la seconde soutenance avec la sauvegarde d'une partie (nous le verrons par la suite, partie 2.10).

Parmi l'implémentation des différentes options et paramètres pour la première soutenance, l'ajout de touches personnalisables a nécessité un travail important. En effet, Unity propose plusieurs manières pour pouvoir interagir avec les touches. Nous avions tout d'abord utilisé la classe "Input" qui permet de vérifier si on appuie sur une touche dans le code source du déplacement. Les inconvénients de cette méthode vont être aux niveaux de la sauvegarde quand on veut modifier les touches ou encore si on veut rajouter la possibilité au joueur de jouer avec une manette.

On a donc décidé d'utiliser l'Input System qui est une méthode plus flexible que l'utilisation de la classe "Input". Mais cette méthode va entraîner des changements importants dans le code sources du déplacement du joueur. En effet, son implémentation est bien différente de celle utilisée précédemment.

Ce système va permettre de définir des actions (sauter, attaquer, marcher ...) ou l'on pourra définir les touches qui leur correspondent.



On peut voir l'action "Walk" qui a deux touches qui lui correspondent, une pour avancer W (Z avec un clavier AZERTY) et S pour reculer

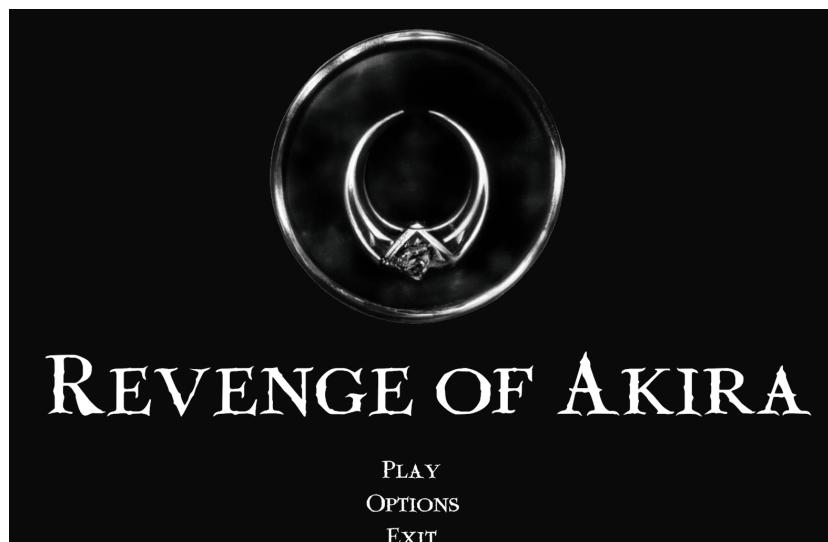
C'est à ce niveau là, que l'on retrouve le système expliqué dans la partie "implémentation de la physique" avec l'exemple de l'axe avant-arrière et en fonction de la touche appuyer, une valeur positive ou négative.

Cette méthode nous permet aussi de mettre pause, de bloquer les déplacements du joueur pour éviter qu'il bouge pendant ses interactions avec le menu.

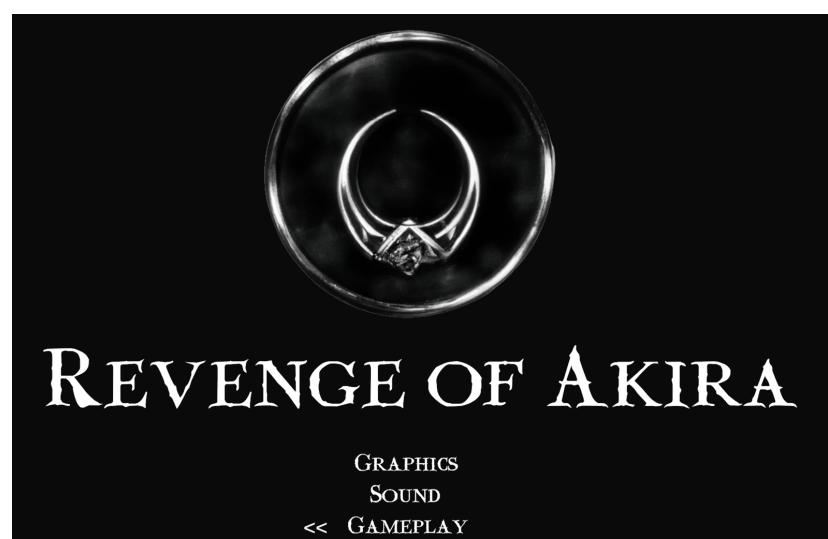
Il sera donc possible pour le joueur de modifier ses options au démarrage du jeu avec le menu principal mais aussi tout au long du jeu avec un menu que nous avons fait pour la deuxième soutenance et qui sera expliquée par la suite(partie : 2.8).

Pour effectuer les interfaces, nous avons créé des “canvas”, des objets graphiques d’Unity qui permettent d’afficher différents éléments au niveau de l’écran. Un canva peut être constitué par différents objets graphiques (bouton, text, menu déroulant ...) qui seront accompagnés ou non de script. Cela permet d’effectuer diverses choses quand on appuie sur un bouton ou encore modifier un texte.

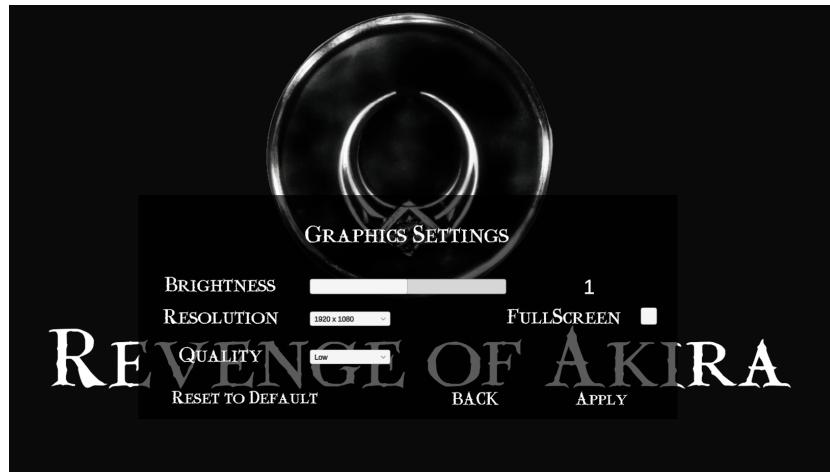
Comme expliqué précédemment, au démarrage du jeu le joueur arrivera sur le canva principal du menu qui permet de lancer une partie, accéder au option et de quitter le jeu. C'est grâce à des boutons qu'on pourra ouvrir les différentes canvas des différentes options, en les activant ou désactivant (sont visibles ou non).



Canva principal du menu



Canva du menu des options



Canva des paramètres graphiques

Plusieurs options sont donc déjà implémentées dans le jeu durant la première période. Il est en effet possible de modifier le volume du jeu, modifier des options graphiques basiques telles que la qualité globale du jeu, la résolution de l'écran et le plein écran mais aussi de les remettre à défaut à des valeurs que nous avons définies. La mise en place de certains éléments a été simplifiée du fait qu'ils étaient déjà des composants proposés par Unity, facilitant leur implémentation au sein des menus, et donc leur facilité d'utilisation par les joueurs.

D'autres éléments ont été ajoutés pour la deuxième soutenance et seront présentés plus tard (partie : 2.8)

Le joueur ayant des attributs comme la vie, un niveau, de l'expérience pour passer les niveaux, nous avons ajouté lors du jeu des indications sur l'état de ceux-ci. Pour cela, on a utilisé encore une fois les "canvases", pour la barre de vie nous utilisons deux bars, de couleur différente (une pour le fond et une pour la couleur de vie). Puis pour les niveaux et l'expérience on l'a fait sous forme de texte. Pour actualiser ces attributs, nous avons dans le code sources des fonctions qui appliquent les modifications sur les attributs puis qui actualisent l'affichage (et qui vérifie pour le niveau si on l'augmente).



Capture d'écran de l'affichage des attributs lors du jeu

Le vert correspond à la vie du personnage, le rouge à ce qu'il a perdu, puis on retrouve le niveau avec l'expérience (expérience actuelle puis le maximum avant de passer de niveau). Cela permettra de tenir le joueur informé sur sa vie actuelle et quel niveau il a atteint.

2.7. Implémentation du son (Mustapha)

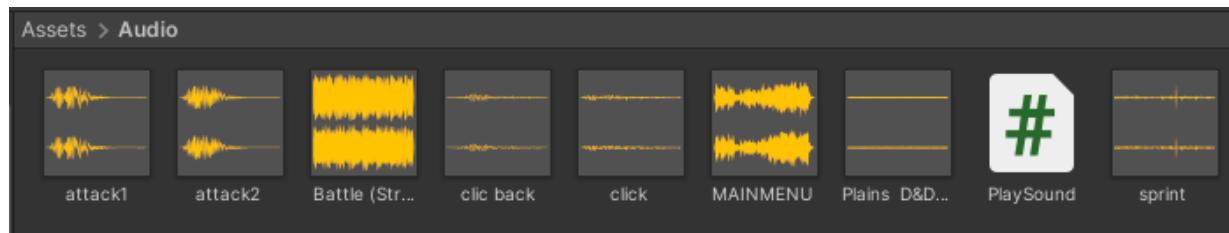
L'implémentation du sons s'est faite en deux parties:
la création du son et son paramétrage dans le jeu.

N'ayant pas le matériel nécessaire pour l'enregistrement audio et pour le sound design, nous avons par conséquent importé des bruitages et des musiques provenant d'internet.

Cependant un des membre du groupe à eu le temps de produire la SoundTrack Principale du jeu que l'on retrouve dans le menu principal. Cette production musicale s'est faîtes à l'aide d'un logiciel nommé "Fruity Loops"



En deuxième partie, l'implémentation du sons de fait à l'aide script qui fait le lien entre l'événement (touche pour marcher, appuyer sur un bouton du menu etc...) et la source audio.

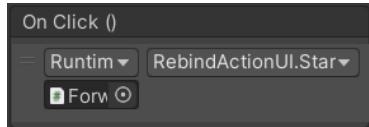


2.8. Finalisation du menu (Thomas)

Pour la deuxième soutenance, nous avons fini le menu principal au lancement du jeu et il restait quelques éléments à ajouter sur le menu que le joueur aura tout au long de la partie. Ces éléments sont en rapport aux multijoueur que nous avons fait juste après la deuxième soutenance et le visuel d'attribut du joueur, ils sont expliqués par la suite (partie : 2.13 et 2.14).

Pour permettre au joueur d'avoir un gameplay agréable et personnalisable nous avons mis en place un système de changement de touches. C'est-à-dire que le joueur pourra personnaliser comme il le souhaite les différentes touches (avancer, tourner, attaquer...).

Pour cela on a utilisé un module de la classe Input System (c'est cette classe que nous avons utilisé pour implémenter les différentes touches) qui s'appelle Rebinding UI. Ce module nous permet d'importer un script dans le projet et c'est grâce à celui-ci que nous avons pu modifier les différentes touches sans trop de difficulté. En effet, nous avons seulement eu à appeler la fonction correspondante au niveau du bouton.



Dans le bouton on choisit ici quelle fonction exécuter lors du clic en fonction de celui-ci

Pour rendre ce système simple et compréhensible pour tout le monde, nous avons indiqué pour chaque contrôle à quoi il correspondait et quelle est sa touche actuelle. Ensuite, il faut appuyer sur le bouton qui correspond à la touche utilisée et une fenêtre apparaît, à ce moment-là la fenêtre va apparaître pour nous dire de presser le bouton que l'on veut choisir. Elle restera affichée tant que nous n'avons pas appuyer sur une touche ou sur un clic de la souris.



Menu des contrôles avec les différents couple (nom de l'action et touche utilisé)

Nous avons également ajouté deux réglages pour le gameplay, le premier est la possibilité de changer la sensibilité de la caméra (qui bouge en fonction de la souris). Pour cela nous avons récupéré une valeur de la caméras, que nous avons utilisé sur le personnage, qui est la vitesse de déplacement de la caméra sur les deux axes que l'on va modifier dans le code sources en fonction de ce que le joueur demande. Etant donné que la vitesse sur les deux axes n'est pas représentée de la même façon (pour la même valeur ajoutée aux deux axes on a pas les mêmes changements de vitesses), nous avons dû trouver un équilibre pour que les changements soit visible mais pas trop dérangeant.

Le deuxième réglage que nous avons ajouté sert à modifier l'axe Y (vertical) pour qu'il soit dans un sens ou dans un autre. Pour cela nous avons procédé à peu près de la même

manière que précédemment, en récupérant une valeur de la cinemachine mais ici qui correspond à une valeur booléenne qui nous dit si on inverse l'axe ou non.



on trouve la barre pour modifier la sensibilité et les boutons pour cocher si on inverse l'axe Y et accéder aux contrôles des touches

Ces nouveaux réglages se trouvent dans le menu lorsqu'on lance le jeu mais aussi quand on met pause au jeu en pleine partie, ce qui permet au joueur de pouvoir modifier son environnement de jeu tout au long de sa partie.

Avec une première partie de sauvegarde effectuée (expliquer plus tard), nous avons ajouté la possibilité au joueur de charger une partie déjà commencée. Pour cela on a ajouté deux boutons qui permettront au joueur de choisir entre commencer une nouvelle partie et charger une partie déjà sauvegardée. Puis un autre bouton lorsque le jeu est en cours pour sauvegarder la progression.

2.9. Création de la deuxième map (Mustapha)

Pour la deuxième map, qui est au passage la map finale, nous avons fait en sorte qu'elle soit beaucoup plus grande que la map précédente pour montrer au joueur que l'aventure ne fait que commencer une fois la map du tutoriel fini.

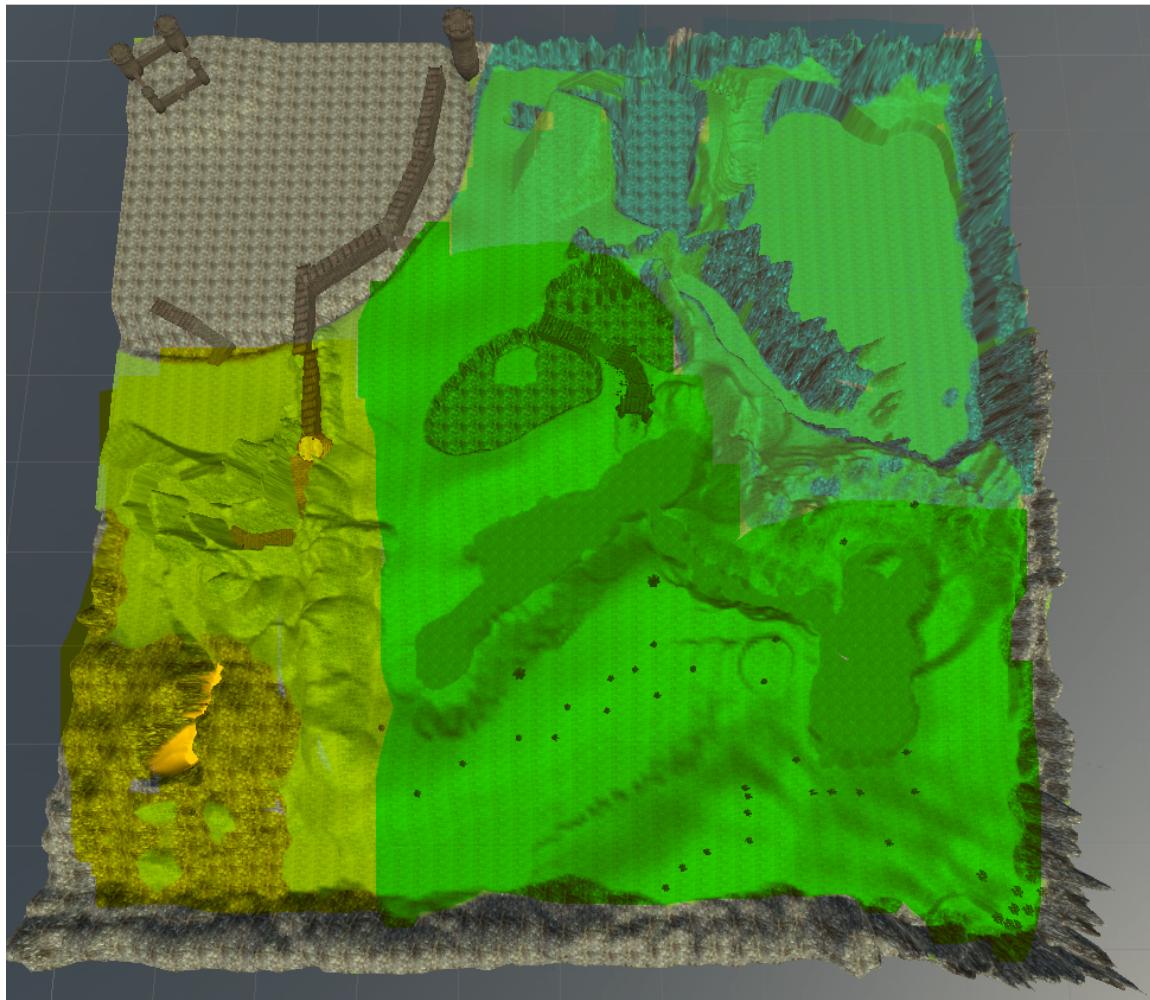
Le point fort de cette partie du jeu est sa capacité à être exploré car elle est très grande et parsemée de lieux divers à découvrir avec comme seul clé pour y accéder, la curiosité. Pour l'instant les textures ne sont pas incroyables mais nous les emploierons par la suite.



point de vue du joueur au moment de l'arrivée

On peut y apercevoir au loin le château qui est entre autres la fin du jeu là où se trouve le boss final

voici la map vu du dessus



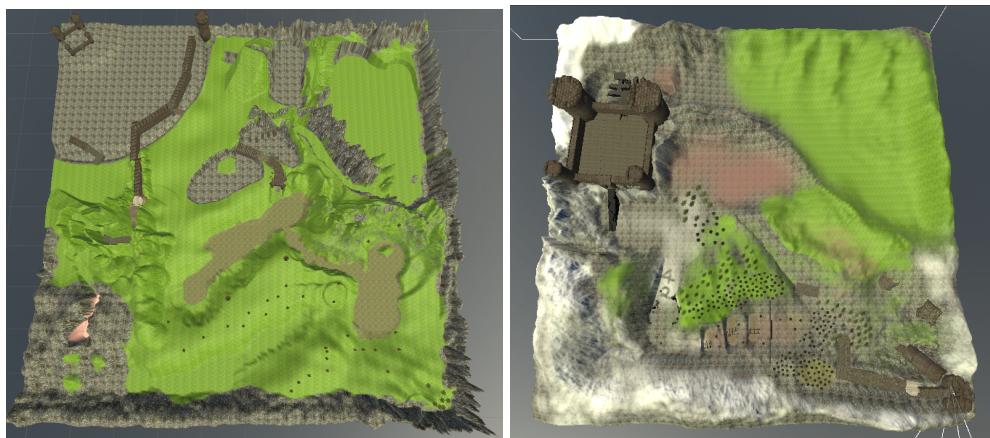
La map est divisée en 4 parties que nous avons mis en évidence avec des marqueurs de couleur sur la screenshot :

- La partie en vert est la partie principale avec des monstres et un pont à traverser, c'est le chemin le plus direct pour atteindre la zone non-colorée où se trouve le château de boss final. C'est aussi la zone la plus vaste et la plus simple à franchir.
- La zone en jaune est labyrinthe et cache de nombreux objets tel que des armes et des potions, elle contient moins d'ennemis et permet de contourner le châteaux et ainsi éviter les monstres les plus difficiles à combattre. Nous rappelons que les trois chemins permettent d'arriver à la zone de fin et totalement possible de revenir en arrière pour continuer l'exploration
- La zone en bleu quant à elle, est une zone très compliquée à trouver de part les reliefs qu'ils l'entourent. Mais une fois dans la zone en question, on va y trouver un boss caché qu'il aussi dur à voir plus dur que le boss final mais une fois ce boss vaincu, le joueur pourra détenir l'arme la plus puissante du jeu car c'est celle qui délivre le plus de dégâts. Ce chemin converge également vers la zone finale qui n'est pas encore développée pour cette soutenance mais qui le sera par la suite.

La Map est munie de mur invisible aux limites des arêtes du carré pour empêcher le joueur de sortir de la map et tomber dans vide.

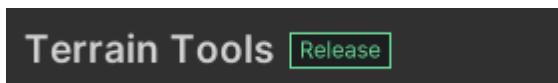
La seconde version de la deuxième map est beaucoup plus fournie, voici les zones principales à explorer en plusieurs étapes.

Voici la map originale et la nouvelle, vue du dessus :

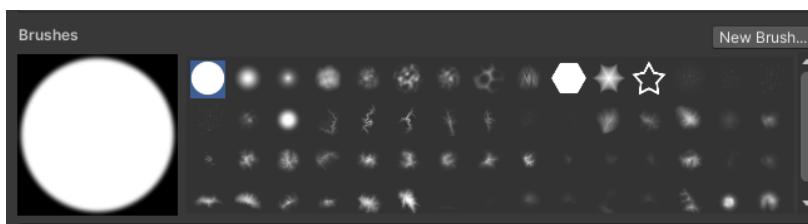


version 1 -> version 2

On peut remarquer que le château prend une place plus importante et que l'environnement est plus réaliste avec une forte présence de la flore et l'amélioration des reliefs et des textures. Les procédés utilisés sont les mêmes pour les deux nouvelles versions, en effet nous avons installé un module interne à unity qui va fournir une palette de reliefs plus importante que ceux fournis de base par unity.



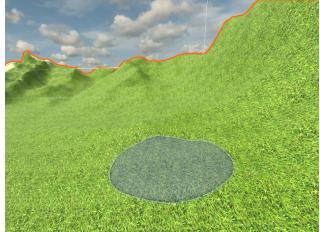
module interne



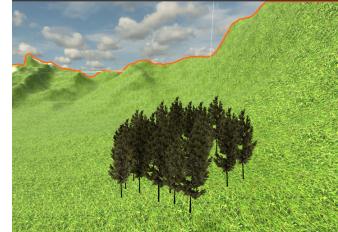
brush disponibles

Pour faciliter le placement des arbres et de l'herbe, des brush sont spécialement adaptés pour en générer à notre guise. Nous pouvons donc également modifier la taille du brush, la densité

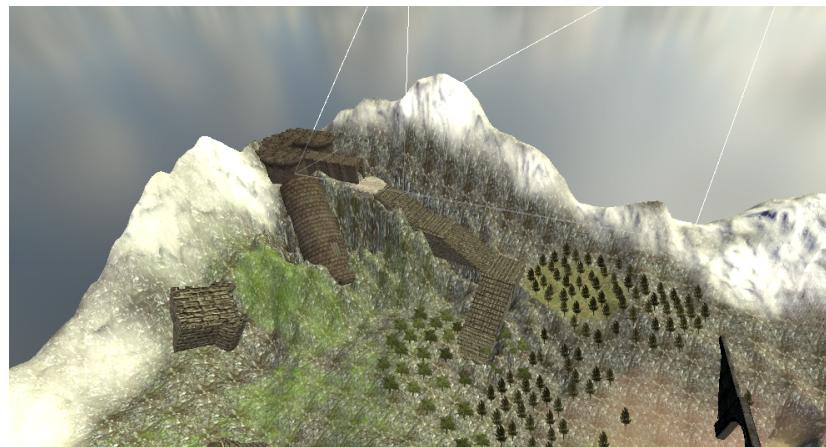
des arbres dans la zone ainsi que la hauteur des arbres, plantes etc... Ce système facilite grandement l'implémentation de l'environnement.



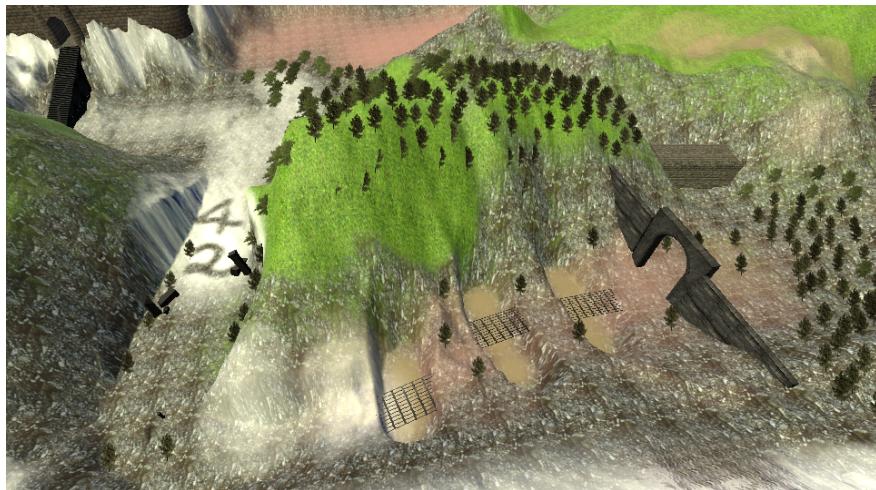
zone de génération génération d'arbres



1) Le joueur apparaît dans une forteresse en ruine et en sortant de la forteresse il pourra profiter d'une vue sur tout le royaume et ce qui l'attend. Cet endroit qu'on appellera le point départ est assez simple à explorer avec quelques items et un chemin toujours guidé par les éléments du décors.



2) La zone 2 est la voie qui nous mène au château. Elle est plus difficile car il faut faire attention à ne pas tomber dans les tranchées et il y a beaucoup de monstres. Si vous n'aviez pas encore remarqué, il y a un easter egg (clic d'oeil caché) en référence à Epita. Après avoir traversé cette zone montagneuse, le joueur aura le choix entre deux chemins possibles. Le premier est celui du château et le second est le boss optionnel .



aperçu du chemin intermédiaire

3) Le Châteaux est la zone la plus dur du jeu avec des monstres surpuissant et une structure labyrinthique, la difficulté est certe plus grande mais les armes et les potion sont plus puissantes, il faut faire attention à ne pas provoquer tous les monstres en même car la partie risque de devenir insurmontable cette partie est appelé “cachots”.



aperçu des cachots

4)Le deuxième choix est en réalité un queu de sac abritant un boss caché qui nous donne l'arme la plus puissante du jeu. Cet endroit est atypique car il n'y a pas de monstres ou d'items et sa teinte rouge rend cet endroit d'autant plus intrigant. Cette zone est pensée pour les challengers. Ce lieu est surnommé “désert rouge” et est totalement désertique.



aperçu de “désert rouge”

4) Pour terminer, la zone finale qui clôture le jeu avec des arches de pierre en ruine et le boss final qui nous attend. Un fois le boss vaincu, la vengeance de Akira sera achevée et le jeu terminé. Le joueur aura la possibilité de retourner sur ses pas pour explorer l'univers à 100 tuer tous les monstres, récupérer toutes les armes du jeu. Au-delà du monde histoire, il y a un mode multijoueur pour plus de diversité.



Pour que le joueur puisse sortir de la map il y a des murs invisibles aux extrémités des bords de la map en appliquant que le box collider et en supprimant les textures.



murs invisibles en vert

2.10. Mise en place de la sauvegarde (Thomas)

La sauvegarde représente une part importante dans un rpg, car c'est des jeux qui sont souvent long à finir où le personnage augmente de niveau et d'attribut, c'est pourquoi tout recommencer a chaque fois n'est pas forcément ce que veut le joueur.

Pour cela, Unity propose différentes choses qui permettent de sauvegarder des données. Tout d'abord on a le PlayerPrefs, c'est une classe qui est relativement sécurisée, facile d'utilisation, et elle est disponible d'office sur tous les supports. Mais elle possède un gros défaut c'est qu'elle est limitée dans le type de données qu'elle peut sauvegarder (entier, chaîne de caractères et float seulement). Comme nous voulons sauvegarder une quantité importante de données pour que cela soit fait une à la fois et même sauvegarder des classes nous n'avons pas principalement utilisé cette méthode.

Nous utilisons cette classe seulement pour enregistrer des données du menu, pour garder les modifications que le joueur fait même après avoir quitté le jeu.

On avait ensuite le choix entre des sauvegardes sous des fichiers en JSON ou en XML, les deux ont le même fonctionnement, seule la manière de représenter les données va changer. En effet, le XML structure les données avec des balises (similaire à un fichier HTML), tandis que le JSON structure les données avec des caractères particuliers « key »: « value ». Comme Unity propose déjà des fonctions qui permettent de sauvegarder des valeurs en JSON et de les lire, nous avons choisi de l'utiliser.

Ensuite, avant de commencer directement il a fallu réfléchir à ce que l'on veut sauvegarder dans une partie, au départ on pensait que l'on pouvait sauvegarder une scène entière ce qui simplifierait le travail, mais cela c'est pas possible sous format JSON et aucun autre format ne permet pas de tout enregistrer. On a dû donc réfléchir à ce qui était important et quels éléments pouvaient varier d'une partie à une autre. Les éléments à sauvegarder sont:

- les attributs du personnage avec ses coordonnées
- le nom de la scène dans lequel le joueur est
- l'attribut de chaque monstres dans les deux map avec leurs positions
- l'état de l'inventaire
- les items présent sur la map

Pour la deuxième soutenance nous avons implémenté les trois premiers éléments. Un problème rencontré assez rapidement avec JSON c'est lorsque l'on veut sauvegarder directement des données de type entier, chaîne de caractères ou float, lors de l'enregistrement des données on retrouve le fichier avec accolade vide sans valeur à l'intérieur. Après quelque recherche effectué et différents test pour comprendre comment cela fonctionne, on a choisi d'enregistrer les données en les mettant toutes dans une classe que l'on va sauvegarder.

La manière la plus pratique d'utiliser le format JSON et de créer des classes qui possèdent toutes les données que nous voulons garder et ensuite d'enregistrer toute la classe entière.

Pour les monstres plusieurs possibilité ont été possible, on a d'abord pensé à utiliser une liste où on met tout les monstres du jeu. La liste composée des attributs des monstres avec leurs

coordonnées nous a amené à quelques difficultés. Bien que l'on ai essayé de différentes manières on n'arrivait pas à enregistrer correctement la liste et se retrouver avec des enregistrement vide, alors que lorsqu'on a essayé de sauvegarder une liste de nombre cela marche bien. Pour résoudre ce problème, on a choisi d'attacher une sauvegarde sur chaque modèle de monstre. On a créé une classe mère Monstre où l'on a implémenter deux fonctions que toutes les classes qui héritent de Monstre définiront. C'est-à-dire que sur chaque modèle de monstre (deux pour la deuxième soutenance) on définit les fonctions pour sauvegarder en JSON et pour lire du JSON.

Pour appeler ces fonctions nous avons utilisé des variables global qui informe si on est en train de lancer une sauvegarde, si c'est le cas on lance alors la fonction sur chaque monstre.

```
public abstract void SaveToJson();
@Override 2 overrides  Thomas LOUP
public abstract void LoadFromJson();
```

Fonctions de la classe mère Monstre que l'on implémente dans chaque classe fille

```
public override void SaveToJson()
{
    ...
}

public override void LoadFromJson()
{
    ...
}
```

Dans chaque monstre on définit les deux fonctions de cette manière

Ensuite, il restait à ajouter un bouton pour que le joueur puisse sauvegarder sa partie. Assez rapidement un problème est survenu, c'est comment faire en sorte de lancer la fonction sur tous les monstres car on doit sauvegarder chaque monstre. On a parlé précédemment d'utiliser une fonction qui vient de la classe mère Monstre, le problème ici c'est comment l'appeler sur tous les monstres en même temps. Tout d'abord lorsque nous avons testé, nous utilisons une touche pour lancer la fonction, mais au passage au bouton il fallait trouver quelque chose pour que toutes les fonctions se fassent à peu près en même temps. Pour cela on a pensé à utiliser une variable booléenne globale que l'on désactive une fois la sauvegarde faite pour ne pas la répéter plusieurs fois. Mais de cette manière après que le premier monstre soit enregistré la variable était redéfinie sur faux. Pour résoudre ça il fallait attendre que tout les monstre soit sauvegarder et ensuite passé la variable à faux. Nous avons finalement utilisé une coroutines pour résoudre ce problème, qui va permettre d'attendre la fin de toutes les tâches sur l'image actuelle (attendre que toutes les fonctions dans Update soit fini) puis mettre la variable booléenne à faux

```

public void ToSave()
{
    IsSave = true;
    StartCoroutine(routine: TimeSave());
}
↳ Frequently called 1 usage ↳ new *
private IEnumerator TimeSave()
{
    yield return new WaitForEndOfFrame();
    IsSave = false;
}

```

code source où on met le booléen correspondant à la sauvegarde à vrai puis on lance la coroutines

On va également utiliser ce système pour charger les données lorsqu'on va lancer le jeu et choisir de charger une partie, ou quand on voudra plus tard sortir du multijoueur et revenir sur la partie seul.

Après avoir rencontré des difficultés pour sauvegarder des listes d'objets, nous n'avons pas implémenté l'enregistrement de l'inventaire et des items, car ils reposent essentiellement sur des listes. Cela a aussi été causé par le temps pris pour l'implémentation du multijoueur que nous détaillerons par la suite (partie : 2.14) et nous avons pris comme décision de prendre le temps qu'il fallait pour le multijoueur (qui est une partie assez complexe) quitte à ne pas faire de sauvegarde d'inventaire.

2.11. Implémentation des premières intelligences artificielles (Tiana Mélanie et Thomas)

Dans notre jeu les intelligences artificielles auront différents rôles, on aura des villageois et des monstres. Les villageois pourront se déplacer aléatoirement dans la map et les monstres nous attaqueront lorsqu'on se rapprochera d'eux. Pour cette soutenance nous avons choisi d'implémenter les monstres car ils nous permettront d'augmenter de niveau et d'avancer dans le jeu (gagner en expérience de combat)

Unity va encore nous aider pour les implémenter. En effet, Unity possède un module **NavMesh**. **Celui-ci va nous permettre de créer une zone sur laquelle l'intelligence artificielle** pourra se déplacer. Ces zones sont créées automatiquement ce qui facilite le travail, mais elles peuvent être modifiées par la suite. Pour notre cas nous n'avons pas eu besoin de les modifier, elles correspondaient déjà à ce que l'on voulait.



image de la première scène avec en bleue la NavMesh des monstres

Pour la deuxième soutenance, nous avons implémenté deux monstres que l'on retrouvera principalement dans les scènes.



à gauche le “Golem” et à droite le monstre “Drake”

Pour choisir ces monstres, nous sommes allés récupérer les modèles 3D sur un site que nous avons déjà utilisé, mixamo.com. Ce site comporte un bon répertoire de modèles 3D de personnages et d'animation. Nous avons également récupéré les animations sur ce site là.

Pour la réalisation du monstre, nous lui avons ajouté un animator qui va fonctionner de la même manière que celui du personnage. En effet, il comporte aussi diverses animations reliées par des transitions.

Ensuite, nous avons réfléchi au fonctionnement du monstre et comment il allait interagir avec le joueur. Pour cela, on calcule la distance entre le monstre et le joueur, en fonction de celle-ci on effectue différentes actions.

La première étape c'est quand le joueur est trop loin, le monstre va alors retourner à sa place d'origine si il n'y est pas et va pouvoir effectuer divers actions (le monstre nommé "Golem" va dormir une fois à sa position initial et le monstre "Drake" va jouer une animation aléatoirement dans le temps). La deuxième étape c'est quand le joueur est proche mais pas assez pour l'attaquer, le monstre va alors le poursuivre tant qu'il est dans cette zone là. Une fois le joueur en dehors de cette zone, la première étape est à nouveau effectuée. Pour mettre une cible à l'intelligence artificielle, il faut donner une destination au monstre, pour cela on donne la position du joueur au monstre grâce à la variable destination. Elle permet de récupérer une position et d'y faire déplacer l'intelligence artificielle.

Une fois que le monstre est arrivé à une certaine distance définie dans ses attributs, on lance l'action d'attaque. Pour effectuer celle-ci on crée un raycast qui permet de voir si le monstre est proche de quelque chose, et si c'est le joueur qui est proche on effectue les dommages correspondant. Pour savoir si c'est le joueur, nous récupérons la tag (permet d'identifier des entités, exemple: player, monster ...) de la cible trouver et nous regardons si c'est player, dans ce cas là on effectue l'attaque.

```
if (hit.transform.CompareTag("Player"))
{
    _PlayerActions.ReceiveDamage(_drakeAttribut.Damage);
}
```

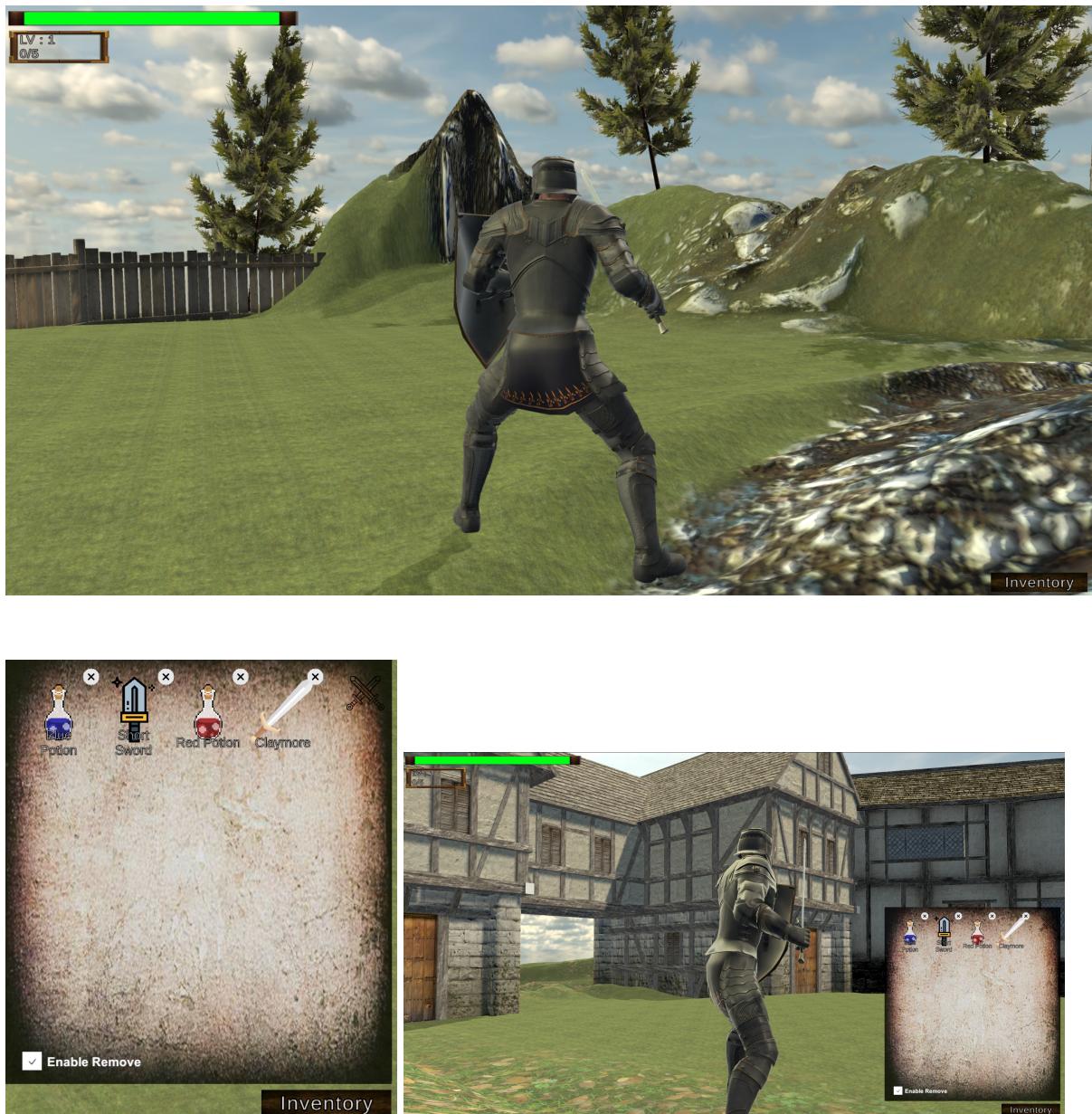
condition qui vérifie si "hit" l'objet proche du monstre est le joueur dans ce cas là on applique les dégâts

Pour donner un peu plus de naturel au combat, le monstre Drake est implémenté avec deux attaques différentes qui se joue de manière aléatoire pour éviter d'avoir toujours la même. Nous avons ajouté cela car c'est le monstre qui sera le plus présent dans la deuxième map, et du côté utilisateur voir toujours la même animation d'attaque peut rendre une image fade du jeu.

A la fin de la deuxième soutenance une grande partie de l'intelligence artificielle a été faite, il ne manquait plus qu'à ajouter d'autres monstres et des villageois pour diversifier les différents éléments dans le jeu.

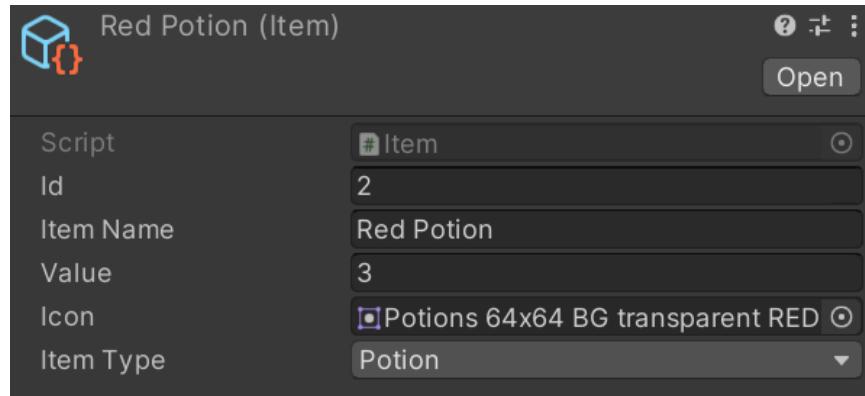
2.12. Améliorations de l'inventaire et des items (Tiana Mélanie)

L'inventaire est une partie essentielle d'un jeu vidéo. A la suite de la première soutenance, l'objectif de la deuxième soutenance était d'afficher chaque items dans l'inventaire. Ci-dessous vous pouvez voir l'affichage de l'inventaire.



Tout d'abord on a dû améliorer le principe de ramassage des objets. Pour la première soutenance, pour reprendre un objet c'était grâce à la touche "E", maintenant il se fait en cliquant sur l'objet à l'aide de la souris.

Pour permettre l'affichage de chaque items dans l'inventaire , nous avons créé une fonction qui prend en paramètre une liste et qui ajoute à chaque fois les items en fonction des attributs qu'on leur a donné notamment le nom, l'icône et la valeur.



un exemple sur le “Red potion”

```
public void ListItems()
{
    //pour éviter les clones.

    foreach (Transform item in ItemContent)
    {
        Destroy(item.gameObject);
    }
    foreach (var item in Items)
    {
        GameObject obj = Instantiate(InventoryItem, ItemContent);
        var itemName = obj.transform.GetComponentInChildren<TMP_Text>();
        var itemIcon = obj.transform.GetComponentInChildren<Image>();
        var removeButton = obj.transform.Find("RemoveButton").GetComponent<Button>();

        itemName.text = item.itemName;
        itemIcon.sprite = item.icon;
    }
}
```

Cette fonction a donc permis l'affichage de chaque item, en prenant leur nom et leur icon respectif.

Par la suite nous avons également mis le bouton “Enable Remove” qui permet d'afficher les boutons “close” lorsque le joueur le souhaite et ainsi faire disparaître l'objet de l'inventaire grâce à la fonction `EnableItemsRemove()`.



Pour finaliser notre système d'inventaire pour la fin de ce projet, nous avons mis en oeuvre une action panel qui possède trois boutons : “use” pour utiliser les potions, “Equip” pour équiper l'épée, et “drop” pour remettre le gameObject de l'item dans la map.



la fonctionnalité :

- lorsqu'on touche sur une potion , seul deux boutons apparaissent : la use et la drop
- lorsqu'on touche sur une épée dans l'inventaire, les boutons Equip et drop apparaissent.

Nous avons donc fait trois fonctions :

- `public void UseActionButton()` : lorsqu'on clique sur le bouton use, il permet d'utiliser la potion et donc incrémenter la barre de vie du joueur.

-`public void DropActionButton()` : lorsqu'on clique sur le bouton drop, il permet de remettre le gameObject de l'item sélectionné sur la map, en face du joueur.

-`public void EquipActionButton()`: lorsqu'on clique sur le bouton equip, il permet d'équiper l'épée du joueur. Il arrive donc directement sur sa main droite.

2.13. Implémentation de différent menu(Thomas)

Pour finir notre jeu, il nous manquait encore quelques interfaces utilisateur (on ne parlera pas ici de celle pour le multijoueur, elle sera expliquée par la suite).

Nous avons donc ajouté un menu lorsque le joueur meurt, un autre pour passer de la première map à la seconde et enfin un menu pour permettre au joueur d'augmenter ses attributs.

Pour le menu qui gère la mort du personnage, nous avons dû récupérer une variable qui indique si le joueur est mort. Si cette variable est vraie, alors on affiche le menu en mettant pause au jeu (pour bloquer toutes interactions avec les éléments de la scène et mouvement).



menu quand le joueur meurt

Sur ce menu, on trouve deux boutons comme on peut le voir sur l'image au-dessus. Le premier “continue” permet de charger la dernière sauvegarde qui à été faite. Le deuxième bouton nous permet de directement rejoindre le menu principal de jeu ce qui peut permettre de lancer une nouvelle partie ou quitter de le jeu.

Le deuxième menu ajouter va nous permettre de passer dans la grande map. Pour que le joueur puisse voir ce menu il devra tuer le premier monstre, pour afficher le menu dans ce cas la nous l'avons simplement activer lors de la fonction de mort du monstre. Ensuite nous avons créé un bouton qui nous permet d'accéder à la deuxième map. Pour cela nous avons chargé la scène puis une sauvegarde pour récupérer les attributs du joueur.

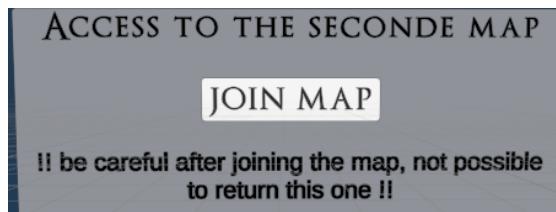


Image du menu pour changer de map

Comme depuis la deuxième map on ne peux plus accéder à la première, nous avons averti le joueur (text en bas de l'image) que s' il appuie sur le bouton il ne pourra pas revenir en arrière.

Enfin le dernier menu ajouter est celui qui permet au joueur de voir ses attributs et de les modifier. En effet, pour notre jeu comme on trouve des monstres plus forts que d'autres nous avions besoin que le joueur puisse devenir plus fort. Pour cela, à chaque montée de niveau le

joueur gagnera un point de skill qui lui permettra d'augmenter ses attributs. Les attributs qu'il pourra modifier seront la vie maximale et la force.



image du menu des attributs

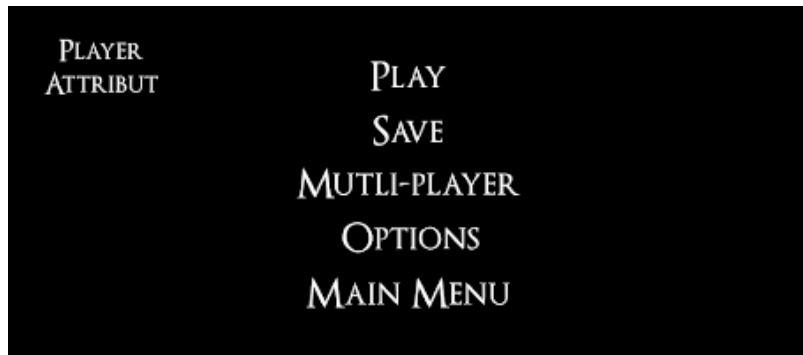
Pour faire augmenter ces deux valeurs nous avons ajouté deux boutons (que l'on peut voir sur l'image ci-dessus) qui sont représentés par un “+”. Lorsqu'on appuie sur ces boutons, une fonction va se lancer pour augmenter les valeurs. Pour la vie (“Max HP”) un point de skill correspond à trois points de vie et pour la force un point de skill a deux points de force. Pour informer le joueur sur les valeurs nous les actualisons à chaque interaction (quand il augmente et quand il ouvre le menu).

Ces trois menus, bien qu'ils soient petits par rapport aux autres, représentent une partie importante dans le déroulement de notre jeu. C'est à partir de là que l'on commence à avoir un jeu sur lequel on peut jouer sans être bloqué dès que le personnage meurt.

2.14. Implémentation du multijoueur (Thomas)

Le multijoueur a été implémenté seulement après la deuxième soutenance à cause d'un retard sur la sauvegarde. En effet, la sauvegarde d'une partie a demandé plus de temps que prévu. Mais étant donné que dans notre jeu ne repose pas entièrement le multijoueur cela n'a pas créé de problème de l'ajouter vers la fin. Pour l'implémenter nous avons utilisé photon qui va nous gérer les serveurs, et Unity propose une grande bibliothèque de méthode avec photon. Photon fonctionne avec des "room", un joueur va soit créer une salle soit en rejoindre une, pour cela Unity propose différentes fonctions qui vont simplifier cela.

Nous avons dû ajouter un bouton dans le menu du jeu qui permet d'accéder au multijoueur



menu que nous avons en cours de jeu, ajout du bouton multijoueur

Nous avons d'abord conçu une première version du multijoueur simple puis une deuxième amélioration que nous expliquerons ensuite.

2.14.1. Première version

Cette première version avait pour but de mettre en place les bases en ce qui concerne le menu du multijoueur. Mais qui devait implémenter la phase du combat dans sa totalité, c'est-à-dire que dans la deuxième version nous avons seulement modifier le fonctionnement du menu qui est essentiel pour du multijoueur.

Pour cette version, nous avons fait un menu où on aura un bouton nous permettant de rejoindre une salle aléatoirement. Une salle correspond à une scène où deux joueurs pourront combattre.

Lorsque le joueur aura appuyé sur le bouton, la scène de combat va se charger c'est à ce moment là que nous allons instancier le joueur. Pour cela il a fallu créer une nouvelle prefab du personnage car on a dû lui ajouter trois composants. Ces composants vont lui permettre d'avoir des informations sur ce qui se passe dans la salle. Mais ils vont surtout servir à partager les informations du joueur avec les autres, ces informations vont être sa position et rotation et enfin les animations. Sans cela, un autre joueur ne verra qu'un personnage fixe qui ne bouge jamais.

Un problème qui arrive assez rapidement c'est la gestion de la caméra. Comment faire en sorte que chaque joueur ait sa caméra. Après pas mal de recherches et de tests effectués nous avons finalement réussi à obtenir ce que nous voulions. Durant cette phase de recherche et de test, un problème fréquent revenait qui était lorsqu'une deuxième personnes se connecter les deux caméras échanger de place. On était sur la caméra du joueur adverse tout en contrôlant notre joueur. Mais cela a été résolu en modifiant l'instantiation des caméras.

Ensuite il a fallu mettre en place le combat, pour cela on a seulement dû modifier une partie du code source du joueur pour qu'il détecte l'autre joueur. Puis pour l'utilisateur on ajoute deux barre de vie, la première pour le joueur et la deuxième pour l'ennemi.

Des difficultés ont été rencontrées ici pour synchroniser les deux, quand le joueur inflige des dégâts à l'autre, celui de son côté n'a pas de changement de vie. Pour résoudre cela, on a d'abord pensé à partager la valeur de la vie par le serveur, mais après plusieurs méthodes utilisées on ne parvenait pas à la faire. On a donc décidé de regarder si le joueur attaque et si c'était le cas on regarde si on touche l'ennemi.

Un problème avec cela c'est qu'on se repose essentiellement sur l'animation, mais pour une raison que l'on ignore et que nous avons pas réussi à trouver, l'animation d'attaque et de saut ne se synchronise pas à chaque fois. Ce qui implique des changements au niveau de la vie en fonction du joueur.

Mais on arrive quand même à avoir un combat entre deux personnes.

2.14.2. Deuxième version

Pour la deuxième version, il a fallu modifier tout le code source pour se connecter au serveur. En effet, on voulait que le joueur puisse jouer avec des amis, pour cela on a dû changer la manière de se connecter et on ne voulait plus rejoindre une salle aléatoirement. On a fait en sorte que le joueur puisse créer une salle ou qu'il puisse en chercher et bien sûr revenir au menu du jeu.



premier menu de la partie multijoueur

Sur ce menu on trouve donc trois boutons pour les actions citées précédemment et une partie (avec le fond rouge) qui permet d'entrer un nom que l'on utilisera ensuite.

Création d'une salle:

Après avoir cliqué sur “Create room” on arrive sur la page permettant la création de la salle. Celle-ci est assez simple, elle ne possède pas beaucoup d’éléments.



page pour créer une salle

On trouve deux éléments, le premier correspond au nom que l'on va donner à la salle et en dessous on trouve le bouton pour confirmer la création. Ce bouton nous connecte automatiquement à la salle que nous avons créé.

Rejoindre une salle:

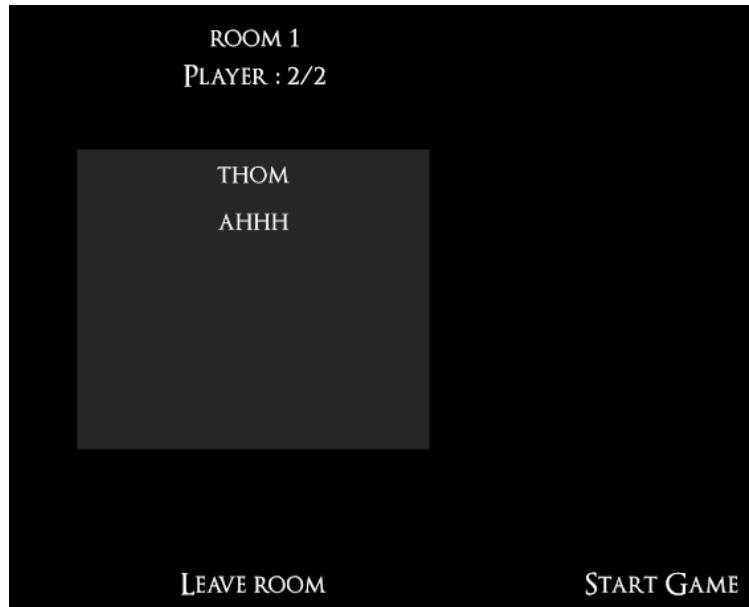
Après avoir appuyé sur “Find room” on arrive sur la page qui va nous proposer toutes les salles disponibles. On pourra alors choisir celle que l'on veut, mais il sera possible qu'on ne puisse pas y entrer si le nombre de joueurs maximum est déjà atteint (nous avons choisi deux joueurs maximum) on se retrouvera alors sur une page d’erreur.



page pour accéder aux salles, ici on a une salle disponible

Sur l'exemple de l'image on a une salle qui à été créée, si on clique dessus on a soit la fenêtre d'erreur qui s'affiche et avertit le joueur que la salle est pleine, soit on entre normalement dans la salle.

Une fois que l'on a rejoint une salle à l'aide des deux méthodes au-dessus on arrive sur une salle d'attente où on peut voir le nombre de personnes dessus et qui y est.



page d'attente de la salle “room 1”

Sur cette image on a différentes informations, on a le nom de la salle, le nombre de personnes et les noms des joueurs qui y sont connectés (nom qui est entré à la première page du multijoueur).

On trouve également un bouton pour quitter la salle et revenir en arrière et un autre bouton pour lancer la partie, c'est-à-dire que l'on va aller sur la scène de combat. Ce bouton n'est pas accessible pour tout le monde, seul le maître de la salle (le créateur si il n'est pas partie) y a accès pour éviter que n'importe qui puisse lancer la partie. Si le maître de la salle part alors la fonction de maître est donnée à une autre personne, mais si il n'y a plus personnes la salle est alors supprimée.

Avec cette nouvelle version, on peut choisir avec qui on veut joueur ce qui est très important dans du multijoueur.

3. Récit de la réalisation

Ce projet à été très intéressant pour chaque membre du groupe et nous à apporter beaucoup de choses. Tout d'abord, cela nous a fait découvrir le fait de travailler dans un grand projet, important et surtout avec une grande durée. Le début n'était pas simple pour savoir comment il fallait s'organiser avec GitHub surtout, avec la mise en place des branches, faire les merges. Par la suite nous nous sommes améliorés sur la gestion de GitHub ce qui a permis de travailler plus efficacement sur nos tâches.

Un point essentiel dans un projet comme celui-ci est la communication, pour cela nous avons créé un groupe discord (permet d'avoir des discussions et channel vocal ce qui est pratique pour un projet numérique).

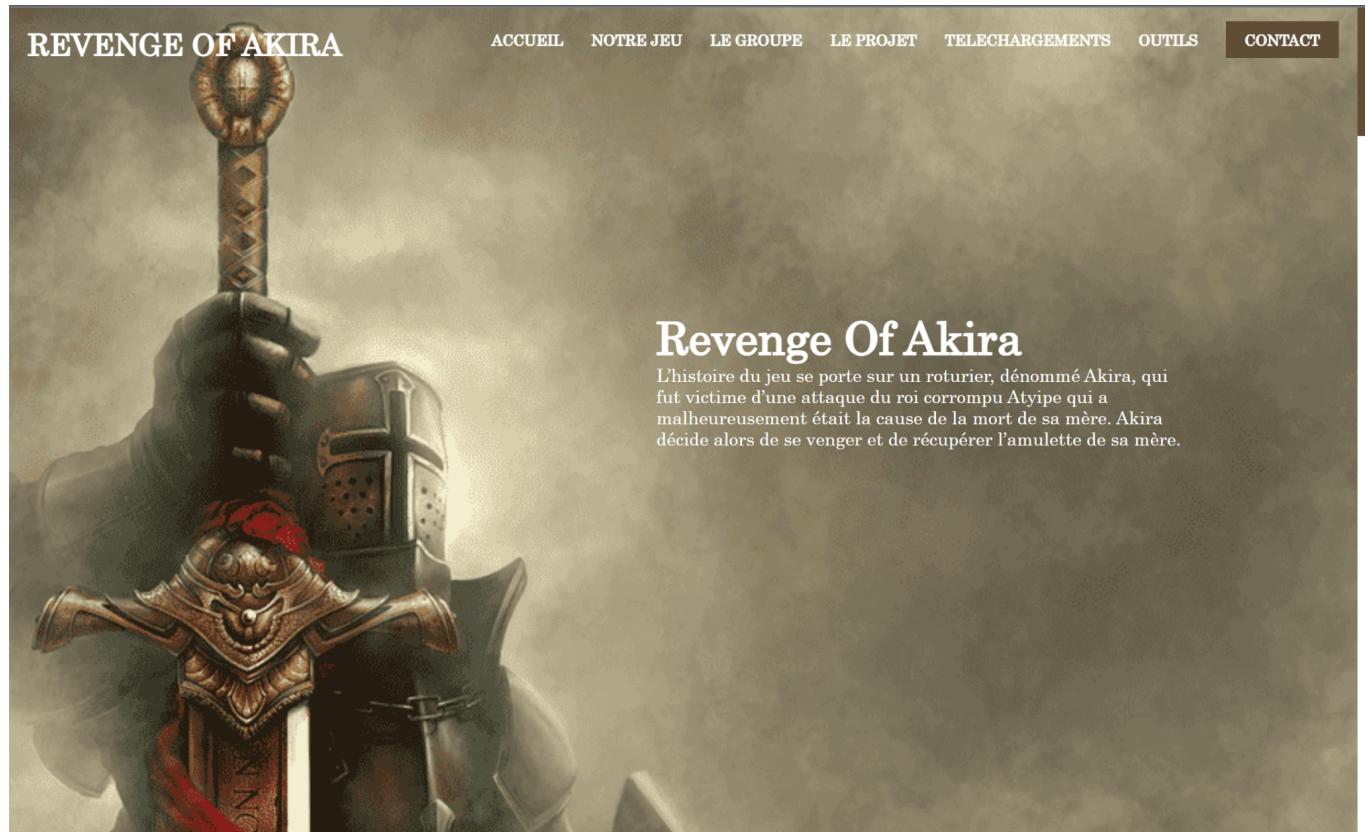
Tout le long du projet, chaque membre du groupe a travaillé avec sérieux, efficacité malgré quelque moment avec moins de motivation, mais cela n'a pas empêché de se remettre ensuite au travail et être de nouveau motivé.

Ce projet a apporté à chaque membre du groupe de nouvelles compétences ou le développement de certaines compétences qui nous seront utiles plus tard pour des projets de groupe mais aussi pour des projets individuels.

4. Industrialisation

4.1. Présentation du Site Web

Dès la page d'accueil, on peut percevoir le personnage principal de notre jeu avec le synopsis de l'histoire. On a choisi de faire un site Web en vitrine.



4.2. Plan du Site Web

- La page d'accueil où l'on retrouve le nom du jeu et le synopsis de l'histoire
- Notre jeu qui contient l'installateur de notre jeu vidéo
- L'onglet présentation du groupe, qui parle de ce dernier et de ses membres.
- Le projet qui est une page qui va expliquer les raisons de notre projet
- La page de téléchargements contient tous les liens voir ou exporter les fichiers en rapport avec le projet avec notamment le cahier des charges et les rapports de soutenance.
- Les Outils qui contient toutes les librairies utilisées pour mener à bien notre projet
- La partie Contact où l'on peut nous contacter

L'image ci-dessous correspond au menu tel qu'il est affiché sur le site web que nous avons développé.

Notre Adresse :
14 Rue Claire Pauilhac, 31000 Toulouse

Envoyer un message

Envoyer

copyright · projet2023 · EPITA *OUMEZIANE Mustapha, LOUP Thomas, RABARISON Tiana Mélanie*. Tous droits réservés

4.3. L'installateur



Afin de créer un installateur facile d'utilisation, nous avons décidé d'utiliser Inno Setup, qui permet de créer, à partir de leur propre langage de programmation (Inno Setup Script) des installateurs facilement utilisables. En indiquant l'emplacement de chaque fichier et dossier nécessaires à l'installation, il crée un exécutable qui fabriquera ensuite les fichiers dans l'ordinateur de l'utilisateur. Le principal avantage de cette méthode est la réduction de place : Revenge of Akira fait environ 1 gigaoctets, mais son installateur ne fait que 250 mégaoctets, ce qui permet donc de l'héberger plus facilement sur le site internet du jeu.

Un script InnoSetupScript est principalement constitué de définition de variables et d'indication de chemins vers les fichiers locaux. La tête du script définit d'abord les caractéristiques de l'application

```

#define MyAppName "My Program"
#define MyAppVersion "1.5"
#define MyAppPublisher "My Company, Inc."
#define MyAppURL "https://www.example.com/"
#define MyAppExeName "Ptojet S2 2.0.exe"
#define MyAppAssocName MyAppName + " File"
#define MyAppAssocExt ".myp"
#define MyAppAssocKey StringChange(MyAppAssocName, " ", "") + MyAppAssocExt

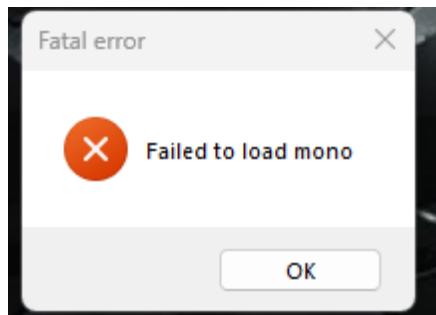
```

Définition temporaire des caractéristiques de VOA

Il est ensuite nécessaire d'indiquer l'emplacement où l'installateur doit se construire
Pour cela on créer un dossier build avec tous les éléments assemblé de notre jeu depuis Unity

 .idea	19/04/2023 13:07	Dossier de fichiers
 Assets	20/04/2023 01:00	Dossier de fichiers
 build	20/04/2023 01:01	Dossier de fichiers
 Documents Projet	19/04/2023 13:07	Dossier de fichiers

cependant, nous avons dû faire face à une erreur jusqu'alors non corrigé et par manque de temps, nous la corrigerons pour la soutenance finales



5. Synthèses personnelles

Mustapha :

Ce projet est une occasion d'exploiter mon imagination et mon ambition. Malgré un manque de temps et d'effectif, j'ai permis à mon équipe d'arriver à un résultat abordable pour une première soutenance notamment en faisant les bonnes concessions et en prenant des décisions importantes car le départ du quatrième membre du groupe a totalement bouleversé la répartition des tâches et les objectifs. Donc un peu de stress mais en prenant tout de même plaisir à concevoir la musique, placer les éléments de la map sans réglementation et bien sûr acquérir de l'expérience dans la conception de jeux vidéo et la manipulation de moteur graphique. Dès le début j'avais une idée bien précise en tête mais trop d'ambition tue l'ambition j'ai du repartir de ce qui était possible de faire dans les temps et j'ai été content d'être à l'écoute d'autres idées et que les autres m'écoutent en retour. La deuxième soutenance était plus mouvementé car beaucoup de problème de communication avec Mélanie, ce manque de communication a créé chez moi une frustration et une peur que cela ralentisse le groupe, résultat nous avons eu du retard au niveau de l'implémentation de l'inventaire et du son, ceci dit, nous avons su en venir à bout et finir le projet.

Thomas:

La création d'un jeu vidéo est pour moi une première, mais c'est un défi très prenant. Du choix du groupe au début du code se passe une partie très intéressante au niveau de la réflexion. En effet, réfléchir à ce que l'on va faire, comment on veut le faire va amener à plein de recherche et d'inspiration sur d'autre jeu, c'est à partir de là que je trouve prenant la conception d'un jeu. J'avais commencé à m'imaginer le jeu dans ma tête, qui au fur et à mesure que l'on construit le gameplay, une image plus claire du jeu se forme. C'est justement cela que je trouve captivant dans la création d'un projet libre, de pouvoir le faire selon nos envies. C'est également très instructif dans la conception d'un projet, l'ordre des choses à faire, le travail de groupe, la répartition du travail, l'organisation du projet et plein d'autres points. Ce projet nous amène sur un point que je trouve très important quand on arrive dans le supérieur qui est la recherche en autonomie. Depuis le début du projet, j'ai effectué un grand nombre de recherches sur les différentes fonctionnalités d'Unity, bien que ça soit pour le projet ça permet aussi de développer nos compétences personnelles.

Mélanie :

La conception d'un jeu vidéo m'a toujours attiré, et ce depuis très jeune, mais je ne m'étais jamais engagée dans un réel projet de cette nature. Le projet de S2 était donc à mon sens, l'occasion de découvrir les outils liés au développement d'un jeu vidéo et des méthodes de travail qui en découlent. Ce projet m'a donc apporté des compétences sur l'utilisation de unity, notamment la physique des jeux, les interfaces, reliées les script adéquat etc ... Git est un outil de collaboration que j'ai également appris durant mon premier semestre mais qui s'est élargi grâce à ce projet, notamment par l'utilisation de git kraken pour voir les différentes élargi grâce à ce projet, notamment par l'utilisation de git kraken pour voir les différentes de mettre en application les compétences apprises en Csharp mais aussi d'en développer d'autres.

6. Conclusion

Le projet Revenge of Akira aura été un réel défi technique, et nous aura demandé beaucoup d'investissement pour arriver à sa forme finale. Tous les éléments, de la création de la carte

au combat final, auront représenté des centaines d'heures de travail, pendant lesquelles nous avons gagné en compétences et appris à utiliser Unity.

Ce projet nous a apporté de nombreuses connaissances techniques et relationnelles. En effet, la création de ce jeu vidéo en groupe nous a apporté des compétences individuelles et collectives. On notera notamment que nous avons progressé en Programmation Orientée Objet (POO) en C et que certains membres du groupe ont également pu améliorer ou mettre à jour leurs compétences en conception et design 3D ainsi qu'en développement web (HTML et CSS). De plus, la réalisation de ce projet était une expérience collective qui nous a permis un développement de soft skills importants. La cohésion, la motivation, l'entraide, la gestion des désaccords et la communication en général sont autant d'enjeux qui ont participé à la réussite du groupe.

Ce projet était donc un avant-goût du monde professionnel de l'entreprise où des tâches techniques sont à accomplir dans un délai imposé et où le respect et l'entraide entre collègues sont primordiaux.

7. Logiciels et outils utilisés

Voici les différents logiciels utilisé et leur fonction dans ce projet:

- Unity (concevoir le jeu)

- GitHub (plateforme de partage et gestion du travail)
- Git Kraken (aperçu visuel de la trace des travaux du GitHub)
- Google Docs (production des rapport de soutenance)
- Discord (communication vocale et textuelle et lieu de réunion)
- FL studio (production musicale)
- Inno Setup (production de l'installateur du jeu)
- Unity Asset store (site d'importation des objets 3D)
- Mixamo (site importation des model 3D et d'animations)
- Photon (platform qui permet l'implémentation du multijoueur)
- Dall-E (génération d'image libre de droit pour notre logo))
- Visual Studio Code (editeur d code pour le site web et pour écrire les script du jeu)
- Youtube et internet en général (acquérir des connaissance)

8. Annexes



premier menu lorsqu'on ouvre le jeu



Image lorsque la première map est lancer