Rapport de soutenance

Charles PROVAIN, Leo PELE, Maxime CHEVRELLE, Tristan MOUNIER

March 2024

1 - Introduction

En vue de valider notre année à l'école de l'Epita, nous devons réaliser un projet jeu vidéo en équipe, composée de cinq membres. Ce projet a pour but de nous faire découvrir la gestion de projet avec tout ce qu'elle implique, le cahier des charges, les délais etc.

Pour nous, ce projet est un moyen de découvrir comment créer un projet de A à Z afin que nous puissions en réaliser de meilleur dans le futur. C'est aussi un moyen d'appliquer nos connaissances vues en cours, de les approfondir voire de découvrir de nouveaux domaines de l'informatique.

Pour ce projet, nous avons décidé de créer un jeu scénarisé d'aventure, de combats et d'énigmes. Le but est de forcer le joueur à la réflexion tout en gardant une certaine tension dans les combats. En créant le scénario du jeu, nous avions pour objectif de captiver les joueurs pour qu'il continue à jouer dans le but de découvrir la fin de l'histoire.

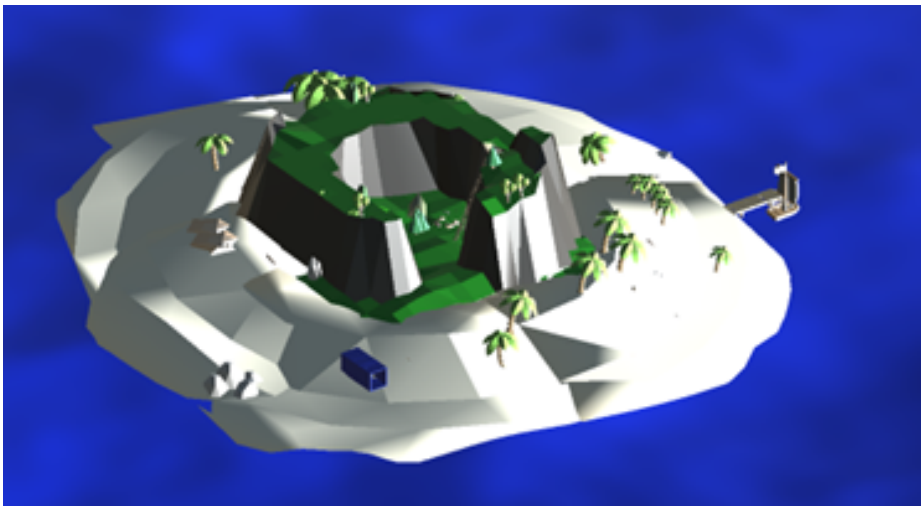
L'univers dans lequel se déroule notre jeu est un monde submergé par les eaux, où seules quelques îles persistent. Durant leur aventure, les joueurs devront explorer cinq îles, chacun ayant sa propre quête et mystère à résoudre. Pour progresser, les joueurs devront affronter des ennemis, tel que les hommes-poisson, tout en faisant appel à leur intelligence pour résolvant des énigmes pour progresser sur la carte.

Chaque joueur joue un rôle unique, que ce soit dans les combats, où le corps à corps et la magie à distance se complètent (Le premier avec un corp fort combattra au corps à corps et le second, avec une intelligence hors norme, combattra avec sa magie à distance.), ou pour la résolution des énigmes.

Dans un monde ravagé par la guerre et la destruction, nos héros doivent survivre tout en découvrant les mystères du monde dans le but de le sauver. Nous vous invitons à plonger dans notre jeu, Nautilus, qui propose un scénario captivant, des énigmes stimulantes et des combats épiques.

Réalisé

Depuis notre dernière discussion, j'ai consacré du temps à la création de la première île sur laquelle nos personnages évolueront. Cette étape m'a conduit à explorer Blender et à exploiter ses capacités pour donner vie à notre vision. J'ai commencé par déterminer la taille de notre île, en m'assurant qu'elle offrait un espace suffisant pour nos aventures à venir. Ensuite, j'ai tracé les contours de l'île, en mettant en place ses plages pour lui donner un aspect naturel et accueillant.



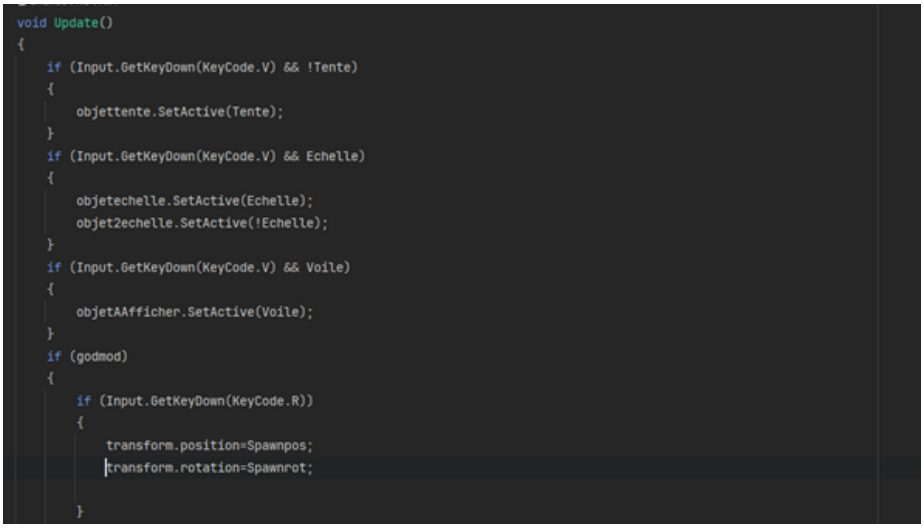
Après avoir mis en place les contours, j'ai créé le centre de notre île, qui sera un lieu clé pour nos deux personnages. Cela m'a permis de concrétiser l'île selon les paramètres que nous avions définis au préalable. Ensuite, j'ai enrichi l'île en ajoutant des détails à l'aide d'assets que j'ai pu trouver sur internet, ce qui lui a donné un aspect plus réaliste. J'ai intégré des éléments tels que des arbres, des pierres, des branches, ainsi que des objets essentiels pour nos héros, comme le bateau, la tente et une échelle pour escalader la montagne. Ces objets feront partir des énigmes afin que les joueurs puissent parcourir les autres îles. J'ai créé un script qui permet aux joueurs d'interagir avec divers objets pour assurer la survie de nos deux héros en quête d'aventure. L'une des premières fonctionnalités que j'ai implémentées est que certains objets ne sont pas visibles

au lancement du jeu, pour correspondre à la situation à la fin de l'aventure. Par exemple, après le naufrage, le bateau qui permettra à nos deux survivants de parcourir le monde est cassé. J'ai donc conçu le script de manière que la voile ne soit pas visible si elle n'a pas été réparée. J'ai ajouté une collision avec le bateau afin de détecté les joueurs aux alentour pour leur permettre de reconstruire la voile lorsqu'ils entrent en contact avec elle.

De même, j'ai mis en place un mécanisme similaire pour une échelle qui permet aux joueurs d'escalader une montagne. J'ai créé deux versions d'échelles : une délabrée et l'autre réparée. Une fois l'échelle réparée, la nouvelle prend place sur le bord de la montagne au même emplacement que l'ancienne. Ensuite, avec la map, je me suis occupée de la réparation du bateau, de l'échelle et de la récupération des matériaux. J'ai également créé un script qui permet aux joueurs d'utiliser l'échelle une fois qu'elle est réparée, leur permettant d'atteindre le sommet de la falaise. Enfin, j'ai reproduit la même logique pour la toile de tente, tel que l'un de nos deux joueurs puissent la récupérer. Ce script a été combiné au script de maxime correspondant à l'inventaire des joueurs. Ainsi, une fois les bons éléments récupérés, les joueurs peuvent les déposer pour reconstruire leur ancien navire et reprendre la mer. Mais avant cela, ils doivent résoudre toutes les énigmes de la première île pour trouver les coordonnées de la suivante.

Bien que la création de la carte ait demandé beaucoup de temps, j'ai trouvé un moyen de contribuer au développement du code. J'ai enrichi le script de Leo ou Maxime en introduisant des nouvelles fonctionnalités de déplacement, tel qu'appelée « Le GODMOD". Cette fonctionnalité permettra aux personnages de traverser les murs et de se déplacer librement à notre guise. J'ai également désactivé la gravité dans ce mode pour permettre aux personnages de rester en place, facilitant ainsi l'exploration rapide de la carte pour présenter notre jeu plus efficacement. Et pour compléter cette fonctionnalité, j'ai ajouté une version spectateur du GODMOD, qui permet aux joueurs de se déplacer de la même manière mais sans voir leur personnage. Cela nous permettra de prendre des photos pour la présentation du jeu. Comme la fonctionnalité du "GODMOD". Cette fonctionnalité permet aux personnages de traverser les murs et de se déplacer librement. De plus, j'ai ajouté une fonctionnalité de téléportation / « respawn » au point de départ du jeu, le ponton,

où nos personnages se sont échoués après le naufrage de leur bateau et de leur équipage.



```
void Update()
{
    if (Input.GetKeyDown(KeyCode.V) && !Tente)
    {
        objettente.SetActive(Tente);
    }
    if (Input.GetKeyDown(KeyCode.V) && Echelle)
    {
        objetechelle.SetActive(Echelle);
        objet2echelle.SetActive(!Echelle);
    }
    if (Input.GetKeyDown(KeyCode.V) && Voile)
    {
        objetAAfficher.SetActive(Voile);
    }
    if (godmod)
    {
        if (Input.GetKeyDown(KeyCode.R))
        {
            transform.position=Spawnpos;
            transform.rotation=Spawnrot;

        }
```

Enfin, j'ai développé la deuxième map qui sert de tutoriel pour les combats, puisque la première carte servait principalement à comprendre les déplacements des personnages et le but du jeu. Sur cette nouvelle île, nous avons ajouté le script d'IA conçu par Tristan pour introduire des ennemis dans le jeu. Ces sbires représenteront les adversaires que nos joueurs devront affronter. La deuxième île se compose de deux plus petites : l'une avec une grande plaine et l'autre avec un volcan. Ces deux îles sont reliées par un pont, offrant ainsi un passage entre elles. Cette disposition permet aux joueurs d'explorer différents environnements et de découvrir de nouveaux défis à surmonter dans leur quête pour avancer dans le jeu.

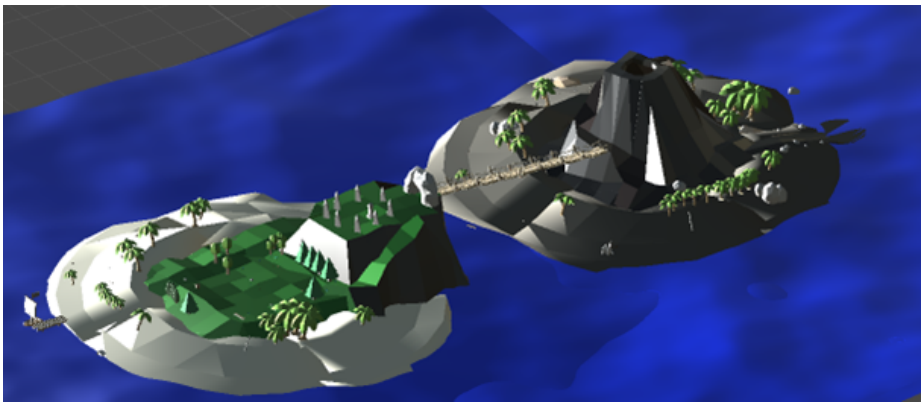
Problemes rencontré

J'ai réalisé mes cartes sur l'application Blender. Mais lors des créations j'ai rencontré plusieurs défis lors de la création de notre jeu. Tout d'abord, la conception des cartes s'est avérée être un processus plus long que prévu, contrairement à ce que l'on pourrait penser, car la vision de notre modélisation peut changer lors du passage de

la carte de Blender à Unity. Comprendre toutes les possibilités et les commandes disponibles sur Blender pour créer l'environnement souhaité a été très long. En passant plusieurs heures à explorer le logiciel, j'ai découvert de nouvelles fonctionnalités et commandes pour affiner ma création. Pour mieux appréhender l'environnement de Blender, j'ai dû consacrer du temps à regarder des vidéos sur YouTube qui expliquaient le fonctionnement de ce logiciel.

J'ai également rencontré des difficultés avec les textures, les couleurs et les objets qui se superposaient mal ou qui apparaissaient ou disparaissaient de manière inattendue dans Unity. En conséquence, j'ai été contraint de recommencer la première carte au cours des premiers jours, car la transition de Blender à Unity ne donnait pas le même résultat. Les problèmes étaient nombreux et trop importants pour être ignorés. Pour résoudre ces problèmes, j'ai dû tester mes cartes à la fois sur Unity et sur Blender afin de comparer les deux résultats. Cela impliquait de passer d'une application à l'autre tout en essayant de résoudre les problèmes rencontrés sur Unity sans en créer de nouveaux. De plus, j'ai trouvé que la gestion de la caméra n'était pas toujours évidente sur Blender et Unity. Parfois, il était difficile de repérer les problèmes potentiels. C'est pourquoi, lorsque j'ai eu accès aux scripts de déplacement du personnage créé par Léo, j'ai pu explorer l'île et remarquer plus précisément certains défauts, tels que des objets flottants ou des zones de terrain qui ne se touchaient pas, laissant des trous dans la carte.

En raison de problèmes de santé survenus en début d'année, j'ai été dans l'incapacité de travailler sur le développement du mode multijoueur. Maxime a donc pris l'initiative de développer ce mode à ma place, afin d'incorporer nos deux joueurs dans le jeu. J'ai décidé de me concentrer sur l'amélioration des scripts de déplacement des joueurs, déjà créés par Maxime et Léo, en y ajoutant des détails pour enrichir l'expérience de jeu. Pour ce faire, j'ai dû consacrer du temps à apprendre le fonctionnement de Unity, tout comme je l'avais fait pour Blender. J'ai visionné des vidéos sur YouTube, consulté ChatGPT pour résoudre certains problèmes spécifiques, et sollicité l'aide de mes coéquipiers dans ce projet pour comprendre les fonctionnalités de Unity, car ils avaient déjà acquis une certaine expérience avec cette plateforme.



Ce qui reste a faire

Pour respecter notre cahier des charges, je vais devoir consacrer plusieurs heures supplémentaires à peaufiner certains détails sur mes deux cartes.Il sera essentiel d'ajouter une multitude de détails pour éviter qu'elles ne se ressemblent trop, notamment car la deuxième île présente actuellement des similitudes avec la première. Pour ce faire, je vais utiliser les nombreux assets que j'ai trouvés au début du projet afin d'incorporer une grande diversité d'objets et de rendre les cartes de plus en plus réalistes, tout en respectant le style "LOW POLY" que nous avons choisi. Le travail le plus long consistera à finaliser toutes les cartes que nous avions envisagées. Cela inclut la création de la troisième carte, qui sera un village nécessitant plus de détails car il s'agit du premier endroit où nous découvrons des bâtiments. Ensuite, je devrai également concevoir la quatrième carte, qui sera le lieu du mini-boss, ainsi que la dernière île où notre aventure prendra fin, c'est-à-dire la cinquième carte. Ce sera le lieu où nos deux joueurs affronteront le boss final pour sauver le monde. Je vais travailler directement avec Unity pour m'assurer du résultat et éviter les problèmes rencontrés jusqu'à présent. En outre, je vais rechercher a créé des morceaux de code pour améliorer encore l'expérience de jeu des joueurs. Si j'achève mes tâches avant les autres membres de l'équipe, je les aiderai sur des tâches où ils rencontrent des difficultés, ou je prendrai en charge des tâches qu'ils n'auraient pas eu le temps de réaliser, comme Maxime

avait eu l'occasion de faire avec le mode multijoueur. Il restera toujours des détails à régler sur les cartes et des fonctionnalités à ajouter grâce à des bouts de code. De plus, je pourrai également m'occuper de la promotion de notre jeu vidéo nommé **NAUTILUS**, que ce soit à travers le site web, Instagram ou d'autres systèmes de communication.

Interface Utilisateur/Deplacement

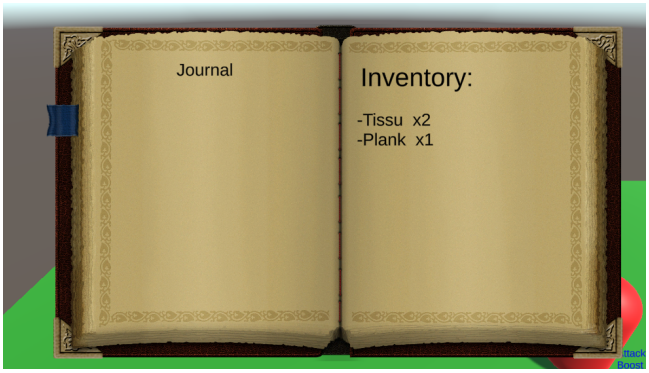
J'ai d'abord commencé a créé un premier script de déplacement avec le transform sur des capsules pour comprendre le fonctionnement de Unity. Par la suite j'ai continué en créant le script pour les attaques du mage. J'ai eu quelques difficultés pour la partie de la définition de la cible.

J'ai commencé par la première interface, le menu de pause. Ensuite, j'ai mis en place le multijoueur. Pour cela, j'ai choisi le plugin qui me paraissait le plus simple, Netcode for GameObject. Pour le faire fonctionner, j'ai dû continuer à travailler sur les interfaces. J'ai alors décidé de créer plusieurs scènes pour les menus, une pour le menu d'accueil et une autre pour le menu de connexion. J'ai rencontré un problème. En effet, le Network Manager était sur le menu d'accueil et je l'activais quand on cliquait sur Play. Le problème était que, en revenant sur le menu d'accueil depuis le jeu et en recliquant sur Play, un nouveau Network Manager était créé en plus de l'ancien. Pour résoudre cela, j'ai dû créer une scène vide au lancement du jeu avec juste le Network Manager qui était lancé et un script qui chargeait immédiatement la scène du Menu. Cela a résolu le problème car cette Scène de Chargement n'était lancée que lors du lancement de l'application. Une fois cela résolu, j'ai

rencontré un autre problème, les caméras des deux joueurs et leur menu pause étaient en conflit. Celui-ci a été plus simple à corriger après quelques recherches sur internet. Le problème suivant a été

de faire spawner des prefab différents pour chaque joueur. J'ai du crée un Player Spawner personnaliser. Mon problème suivant a été

pour le premier sort que j'avais réalisé. Je n'arrivais pas à définir sa position. Au final j'ai compris que la cible était définie pour le client et que le server, au moment de spawn l'objet, n'avait pas de cible. J'ai par la suite continué à faire les compétences du mage pour ensuite faire celle du guerrier. Pour cela chacun a un script attacher qui permet de les gérer. Puis j'ai commencé à faire l'UI ingame afin d'avoir les infos sur les temps de recharge des compétences. En continuant sur les interfaces, j'ai fait une première version du journal. Ensuite j'ai commencé les inventaires. Pour cela j'ai voulu utiliser les collisions avec les objets pour les détecter et les mettre dans un dictionnaire. J'ai créé un enum avec tous les type d'item comme clé du dictionnaire associer au nombre présent dans l'inventaire. Lors des tests, je me suis rendu compte que le joueur passait à travers les objets. Après des recherches je me suis rendu compte que cela venait du déplacement avec le Transform, j'ai donc changé pour utiliser le rigidbody.



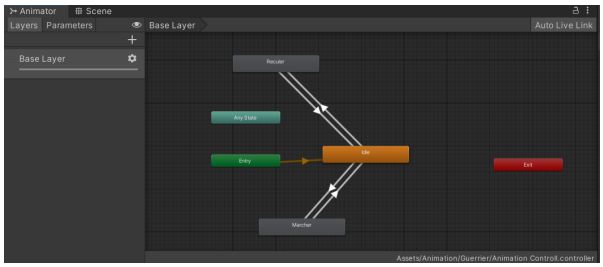
Ensuite, avec la map, je me suis occupée de la réparation du bateau, de l'échelle et de la récupération des matériaux. En faisant cela, j'ai rencontré un problème du a l'héritage. En effet, Pour hériter de la class NetworkBehaviour, il faut un network Object Component et Unity le met tous seul. Cependant pour les interactions avec le bateau et la tente, j'ai décider de passé par une class abstraite, du coup Unity n'as pas rajouter le network Object Component et envoyait une erreur difficile à comprendre. Ensuite je me suis occupé du changement de scène entre les différentes iles.

Animation

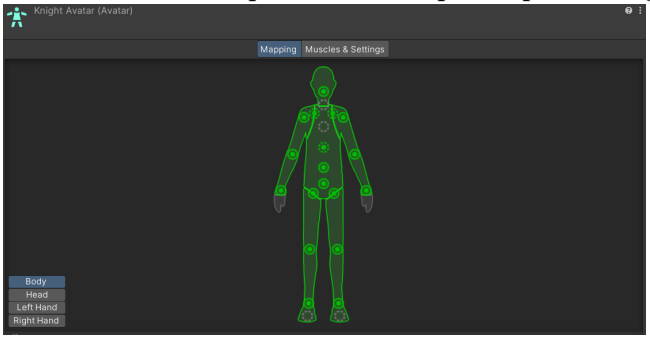
L'animation dans Unity repose sur le système d'Animator, qui permet de gérer les états d'animation et les transitions entre eux. Avec l'Animator Controller, les développeurs peuvent orchestrer des séquences d'animations complexes, en contrôlant les paramètres qui déclenchent les différentes animations.

L'Animator Controller permet d'organiser et de maintenir un ensemble de clips d'animation et les transitions associées. Les clips utilisés ici sont :

- "Idle", correspondant à l'état par défaut (quand il n'effectue aucun mouvement ni action)
- "Marcher", quand le personnage se déplace tout droit
- "Reculer", quand le personnage recule

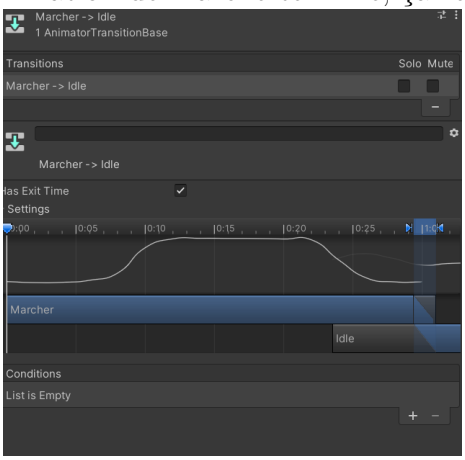


Il est important de bien configurer le personnage, en le définissant comme humanoïde, il faut aussi vérifier que ce personnage possède un Avatar (squelette) puisque c'est grâce à cela que Unity va reconnaître les différentes parties du corps du personnage.



Pour configurer l'Animator Controller au personnage, j'ai ajouté un composant Animator dans lequel je vais associer l'Animator Controller. Dans le script du personnage, j'ai créé une variable An-

imator qui va être le composant que je viens d'ajouter. Afin de gérer les transitions entre les différents états d'animation, plusieurs paramètres (de types booleen, int, float,. . .) peuvent être ajouté. Ces paramètres peuvent être modifier dans le script associé a l'animator controller, ici, j'ai associé l'animation " Marcher " à " Idle ". Un fois l'animation de marche terminé, ça revient à l'état initiale " Idle ".



Une fois tout cela fait, le personnage est prêt à jouer ses animations au bon moment.

son

Dans un jeu vidéo, le son contribue à l'immersion du joueur. Il permet de procurer des sensations, en utilisant l'ouïe pour rendre le jeu plus vivant et plus immersif. Unity possède plusieurs composants audios, ce qui facilite l'implémentation de l'audio. Pour cette première soutenance, j'ai utilisé le composant Audio Source. Il permet de jouer différents types de clips audios. J'ai configuré le son pour quand le personnage se déplace et quand le mage tire un projectile, j'attends l'implémentation des autres fonctionnalités avant de tout ajouter. Les sons déjà ajoutés seront amenés à changer par la suite.

Pour implémenter le son au personnage, j'ai ajouté dans son script des variables Audioclip, ces variables vont être nos clips audios. Une variable AudioSource va nous permettre de jouer le clip audio. Puis il ne reste plus qu'à associer nos clips audios aux actions que nous voulons leur associer.

Pour la marche, j'ai vérifié qu'il n'y avait pas de clip audio en cours d'utilisation. Cela évite de recommencer le clip audio en boucle lorsque nous restons sur la touche pour avancer. Plusieurs paramètres audios sont mis à disposition, j'ai utilisé le pitch pour régler la vitesse du clip afin de l'accorder avec l'animation.

Graphisme

Les graphismes sont un élément essentiel des jeux vidéo, elle offre une esthétique visuelle ainsi qu'une expérience immersive pour le joueur.

Dans ce projet, nous sommes parti sur un style en low poly. C'est un type de maillage polygonal ayant un nombre faible de polygones. Ce qui permet d'avoir des textures et modèles simples et peu détaillés.

0.1 Personnage

Dans notre projet, nous voulions 2 personnages : un guerrier et un mage.

J'ai d'abord essayé de modéliser un personnage sur Blender en utilisant les outils mis à disposition et en suivant des tutoriels. Cependant, le résultat finale n'était pas celui attendu et j'ai pu rencontrer plusieurs problème lorsque qu'il fallait faire l'armature du personnage. J'ai préféré me concentrer sur la partie du script de personnage que sur la partie visuelle. C'est pourquoi j'ai récupéré deux textures de personnages, un pour le guerrier et un autre pour le mage sur le site : Poly Pizza

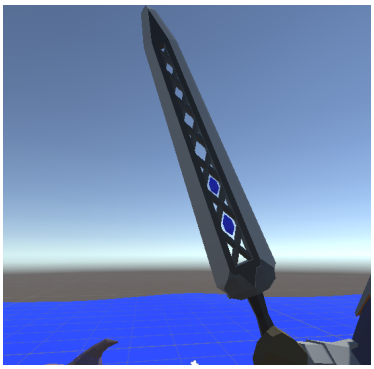

Ces assets de personnages correspondent bien au style graphique Low Poly et restent dans le style du guerrier et du mage. En important ces personnages dans Unity, j'ai rencontré quelques problèmes, notamment avec le mage qui n'avait pas de texture. En modifi-

ant quelques paramètres, j'ai fini par retrouver sa texture initiale. Le problème suivant a été de faire apparaître des prefabs différents pour chaque joueur. J'ai dû créer un Player Spawner personnalisé. Les deux personnages possèdent un script comprenant les mouvements, les animations, et le son. Ils ont la possibilité de se déplacer librement, et d'interagir avec certains objets présents dans l'environnement. Pour les déplacements, j'ai utilisé le composant " transform ". Ce composant permet de définir la position, la rotation, et l'échelle d'un GameObject. Ici, notre personnage.

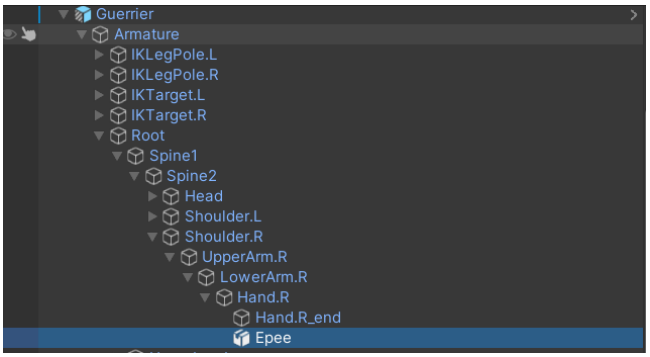
```
void Update()
{

    if (Input.GetKey(Avancer) && !Input.GetKey(KeyCode.LeftShift))
     {
         transform.Translate(0, 0, VitesseDeMarche * Time.deltaTime);
         animator.Play("Marcher");

         if (!audioSource.isPlaying)
         {
             audioSource.pitch = 0.75f;
             audioSource.clip = Marcher;
             audioSource.Play();
         }
     }
    else
    {
        if (audioSource.isPlaying && audioSource.clip == Marcher)
        {
            audioSource.Stop();
            audioSource.pitch = 1;
        }
    }


     if (Input.GetKey(Reculer))
     {
         transform.Translate(0, 0, -(VitesseDeMarche / 2) * Time.deltaTime);
         animator.Play("Reculer");
     }


     if (Input.GetKey(Gauche))
     {
         transform.Rotate(0, -VitesseRotation* Time.deltaTime, 0);
     }

     if (Input.GetKey(Droite))
     {
         transform.Rotate(0, VitesseRotation * Time.deltaTime, 0);
     }

     if (!Input.GetKey(Avancer) && !Input.GetKey(Reculer))
     {
         animator.Play("Idle");
     }


}
```

0.2 Items : Epée et Baguette

Ces deux personnages possèdent leur propre item. L'épée pour le guerrier et une baguette pour le mage. Ces deux modèles proviennent du site : Sketchfab, ils restent dans le thème low poly et son en raccord avec les couleurs des personnages.



J'ai associé chaque items au personnage. Pour ce faire, j'ai mis le modèle dans la branche de la main droite (pour le mage et guerrier)



Cela permet que, quand les personnages sont en mouvement, les items restent fixés à eux, comme s'ils étaient directement intégrés. La baguette possède un script, car il faut pouvoir l'utiliser. Le script reste simple : lorsqu'un clic gauche de la souris est détecté, un laser est instancié à partir d'un préfabriqué. Il se déplace dans la direction du vecteur LaserSpawn, qui va être le bout de la baguette, avec une vitesse donnée en paramètre.

```
public class Baguette : MonoBehaviour
{
    public Transform LaserSpawn;
    public GameObject LaserPrefab;
    public float LaserVitesse;



    public AudioClip Laser;
    public AudioSource AudioSource;



    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            var laser = Instantiate(LaserPrefab, LaserSpawn.position, LaserSpawn.rotation);
            laser.GetComponent<Rigidbody>().velocity = LaserSpawn.forward * LaserVitesse;
            AudioSource.clip = Laser;
            AudioSource.Play();
        }
    }
}
```

Intelligence Artificielle

Pour cette soutenance, l'objectif était de mettre en place l'IA des sbires. Pour cela, j'ai réalisé deux scripts, un qui permet à l'IA de prendre des informations sur le monde, que l'on va appeler "AISensor", commun à tous les ennemis, et l'autre qui s'occupe des animations et déplacements de chaque ennemi, nommé "AIMovement", qui est donc propre à chacun. Le principe est simple, chaque ennemi a sa propre vision et portée d'attaque, et 2 états sont disponibles sur chaque monstre, ce sont des booléens publics qui vont s'actualiser à chaque frame, que l'on va utiliser dans d'autres scripts.

• Soit l'ennemi ne voit pas le joueur, c'est à dire que le joueur est plus loin que la vision du monstre, dans lequel cas l'ennemi ne fait rien.

• Soit l'ennemi voit le joueur mais ne peut pas l'attaquer, c'est à dire que le joueur est plus loin que la portée d'attaque du monstre mais moins loin que sa vision, dans lequel cas nous mettons le

booléen Poursuite à True.

• Soit l'ennemi voit et peut attaquer le joueur dans lequel cas nous mettons le booléen Attaque à True.

Maintenant, pour détecter le joueur, j'ai d'abord pensé à initialiser le joueur comme un public transform pour que l'on renseigne le joueur en question directement dans Unity, mais le problème est que l'on ne peut que détecter un seul joueur avec cette technique, ne marchant pas pour le multijoueur. J'ai donc créé une méthode permettant de trouver l'ennemi le plus proche parmi tous les objets du jeu avec le tag "Player", un tag que l'on a mis sur nos deux joueurs. La méthode retourne le joueur qui est le plus proche du monstre : la position de l'ennemi se notant "transform.position", on trouve la distance entre le joueur et l'ennemi avec un "go.transform.position - transform.position;", go étant un élément du tableau de tous les objets avec le tag "Player", et on retourne l'élément qui a la plus courte distance avec l'ennemi.

```
public GameObject FindClosestEnemy()
{
    GameObject[] gos;
    gos = GameObject.FindGameObjectsWithTag("Player");
    GameObject closest = null;
    float distance = Mathf.Infinity;
    Vector3 position = transform.position;
    foreach (GameObject go in gos)
    {
        Vector3 diff = go.transform.position - position;
        float curDistance = diff.sqrMagnitude;
        if (curDistance < distance)
        {
            closest = go;
            distance = curDistance;
        }
    }
    return closest;
}
```

Modèles des deux sbires

Ce premier sbire à gauche est le sbire mêlé, et ces deux modèles sont en poly art car c'est le style que nous avons choisi pour notre jeu. Ces deux modèles et images viennent de l'Asset Store de Unity dont vous trouverez les liens sur notre site. Le sbire à droite est celui qui attaque de loin, mais il est important de noter que ce ne sont pas les modèles définitifs de nos sbires.

sbire de mêlé

sbire à distance

Le second script reprend la logique du premier, il va prendre la valeur des booléens que renvoie le premier script pour y appliquer l'IA divisée en trois états sur l'ennemi ; j'ai fait le choix de séparer cela en deux fichiers car chaque ennemi possède des animations différentes, donc au lieu de faire deux scripts volumineux se ressemblant, j'ai fait le premier script en commun pour les deux monstres et le script de l'IA pour chacun des monstres.

Pour déplacer l'ennemi vers le joueur, j'utilise le composant de Unity appelé "Nav Mesh Agent" qui permet de déplacer le monstre jusqu'au joueur en prenant le chemin le plus court et en contournant les obstacles, comme des arbres et des pierres. Ce composant rajoute la possibilité de choisir la destination de l'ennemi (agent.SetDestination()) ainsi que le droit de bouger ou non (agent.isStopped = false), cela avec une vitesse modifiable dans Unity. Ainsi, j'ai codé l'IA de cette manière;

- Si le booléen "Attack" dans AISensor est vrai, alors agent.isStopped

est vrai et l'ennemi fait son attaque.

• Si le booléen "Poursuite" est vrai, alors agent.SetDestination(Player.position) va faire poursuivre l'ennemi vers le joueur sans oublier de mettre agent.isStopped à faux.

• Sinon, si aucune des ces conditions sont vérifiées, on en déduit que l'ennemi est à l'état "Idle" et on va simplement indiquer que agent.isStopped est vrai.

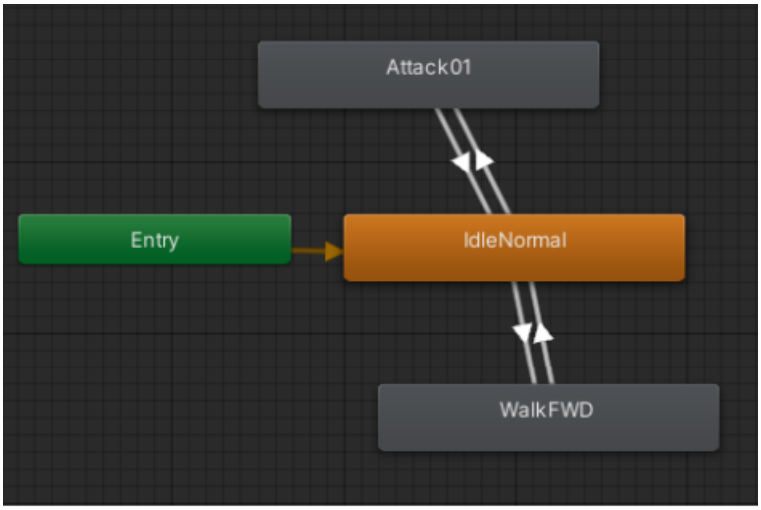
Problèmes rencontrés

Durant la création de l'IA je voulais instaurer un nouvel attribut exclusivement pour les sbires qui attaquent de loin, étant que si le joueur s'approche trop prêt du monstre, il recule dans la direction opposée puis continue d'attaquer. Je voulais donner un air un peu plus réaliste à ces sbires pour qu'ils soient un peu plus durs à battre. J'ai alors créé un nouveau booléen publique dans AISensor qui devient vrai quand le joueur est à 4 unités de distance de l'ennemi ou moins (au hasard), puis quand il devient vrai l'ennemi part à une distance de 4 unités du joueur. Simplement, quand j'essayais mon programme, soit le monstre partait dans le mauvais sens, soit il s'arrêtait avant de pouvoir être dans sa zone d'attaque. Après beaucoup d'essais ratés j'ai alors abandonné cette idée pour cette soutenance mais instaurer cela sur un boss peut être intéressant. Je n'ai pas trouvé de solution à ce problème par faute de temps mais je n'hésiterais pas à demander de l'aide pour l'avenir.

Nous avons ainsi respecté la partie de l'IA dans les temps du cahier des charges et nous avons prévu de nous attaquer à la prochaine soutenance à l'IA du boss et du mini-boss.

Animation-Ennemis

Pour les animations, l'objectif pour cette soutenance était de mettre en place les prototypes d'animation des sorts, attaques et déplacements de notre personnage ainsi qu'aux ennemis. M'occupant également de l'intelligence artificielle des ennemis, j'ai trouvé que c'était une bonne idée de m'attaquer également à leurs animations. À noter que les animations des monstres étaient déjà disponibles sur le Asset Store, ce qu'il fallait faire là était d'enclencher les animations au bon moment. Comment faire? Avec le composant Animator de Unity, il est possible de jouer des animations simplement.



Voilà à quoi ressemble le composant, les 3 blocs "Attack, Idle et Walk" sont 3 animations du monstre, et l'ennemi fait de base l'animation que le bloc Entry pointe, donc Idle. Pour passer d'une animation à une autre, de Idle à Walk par exemple, il est possible sur ce composant de créer des conditions pour jouer une animation (représenté par des flèches), j'ai donc créé deux booléens, Attack et Walk, et imposé simplement que l'animation d'attaque se lance si Attack est vrai et Walk est faux, et inversement pour Walk. Il n'est pas nécessaire de faire de conditions pour Idle.

J'ai ensuite modifiés ces booléens directement dans mon dossier de mouvement de monstre pour l'intelligence artificielle, car le comportement du monstre est lié à ces animations. En effet, j'avais énoncé trois états pour l'IA des ennemies, étant les mêmes que les animations. Voici donc à quoi ressemblerait mon script pour les animations :

```
public class AIMovementM : MonoBehaviour
{
    public AISensor _sensor;
    private Animator animator;

     void Start()
    {
        animator = gameObject.GetComponent<Animator>();
    }


    private void Update()
    {
        if(_sensor.CanAttackPlayer)
        {
           animator.SetBool("Walk", false);
           animator.SetBool("Attack", true);
           return;
        }

        if (_sensor.CanSeePlayer)
        {
            animator.SetBool("Attack", false);
            animator.SetBool("Walk", true);
            return;
        }
        animator.SetBool("Walk", false);
        animator.SetBool("Attack", false);
    }
}
```

Nous avons ainsi respecté la partie de l'animation dans les temps du cahier des charges et nous avons prévu de nous attaquer à la prochaine soutenance à la finalisation de ces prototypes d'animations et bien sûr aux animations des futurs ennemis tels que le boss et le mini-boss.

Script

Pour le script, l'objectif pour cette soutenance était de faire le prototype de trailer et cinématique du jeu, les dialogues avec les PNJ et le prototype d'alphabet. Seulement, nous n'avions pas pensé à la cohérence de ces avancements en janvier, puisqu'il est difficile de proposer un trailer et une cinématique quand nous n'avions pas encore tous les éléments du jeu et seulement la première map de finalisée. De plus, les PNJ n'apparaissent qu'à la quatrième île du jeu. En addition, nous avions discuté en groupe et admis qu'un alphabet ne serait pas nécessaire puisque nous proposerons des pancartes avec seulement des symboles. J'ai donc préféré me concentrer sur l'IA et les animations plutôt que le script, ce qui fait que nous avons du retard sur le script par rapport au cahier des charges. Je prévois ainsi de faire le trailer et la cinématique pour la prochaine soutenance, ainsi que le dialogue avec les PNJ.

Nous n'avons ainsi pas respecté les promesses du cahier des charges du script dû à des problèmes de cohérence et nous prévoyons de faire la cinématique et le trailer à la dernière étape du développement de notre jeu, puisque nous aurons tous les éléments pour les faire.