

Léo FARHI (Chef du projet)
Hugo RUBIO
Geoffrey ELBAZ
Léo ALBA SANCHEZ

Rapport de soutenance n°2

Regain The World !

RELIK

30 avril 2021



Sommaire

Introduction	3
Rappel du projet	3
Rappel du dernier rapport	3
Level Design	3
Intelligence Artificielle	4
Présentation générale du projet et des tâches individuelles	5
Avancement général du projet	5
Présentation du planning de réalisation	5
Level design	6
Constructions	6
Générateur de PNJ avec tenue semi-aléatoire	7
Système d'intelligence artificielle	11
Les déplacements	12
Les combats	13
Les animations	15
Musiques	16
Network	16
Optimisation	17
Modèles 3D	18
Déduction	19
Avancement général du site Web	19
Problèmes rencontrés	19
Réalisation	19
Entente du groupe	20
Bugs et solutions	20
Conclusion	20
Rappel sur ce qui est fait	20
Rappel sur ce qui est à faire	21
Annexes	22
Images du jeu	22
Site Web	26

I. Introduction

Rappel du projet

Cette partie est un rappel de ce qu'est "Regain The World".

Le groupe RELIK a été créé le 2 décembre 2020, il est composé de 4 personnes (Léo FARHI , Hugo RUBIO , Léo ALBA SANCHEZ et Geoffrey ELBAZ) très intéressées par la création et le fonctionnement d'un jeu vidéo et qui ont alors commencé à développer un jeu pour le projet .

Ce jeu est nommé Regain the world et sera un jeu Aventure et Open World du genre Heroïc fantasy, jouable en solo et en multi, où le but de nos personnages sera de regagner leur monde (dit réel). Le type, le style graphique, l'environnement, les mécaniques, le scénario du jeu sont inspirés de beaucoup de jeux et d'univers fantastiques (Zelda, Genshin Impact, les premiers tomb raider, ...)

Rappel du dernier rapport

- Level Design

Un jeu de type vidéoludique est aujourd'hui apprécié pour sa complexité, son scénario ou sa beauté. Le level design est alors une partie importante du jeu car c'est en partie elle qui lui permet de prendre vie et de choisir sa forme. Si l'on en s'en remet rapidement à la philosophie, le level design forme la peau et la chair qui permettent la confirmation de son existence par les autres car vu et reconnu par ces derniers.

Nous voulons un jeu attractif qui donne envie de jouer, c'est pour cela qu'il fallait que le nôtre soit "beau" et c'est là que cette partie prend toute l'importance qu'on lui consacre. Car notre jeu doit taper dans l'œil du joueur et lui donner envie d'y jouer. Pour cela il fallait créer plusieurs "régions" ou "lieux" attractifs et tous différents tout en gardant une certaine logique pour la suite de l'histoire. Ces lieux permettent à la fois au joueur de se situer et de délimiter son terrain de jeu ainsi que celui de IA dont on parlera juste après.

- Intelligence Artificielle

L'IA prend une partie assez conséquente du fonctionnement du jeu. En effet, celle-ci concerne l'implémentation des personnages non joueurs (PNJ), des ennemis et des personnages principaux remplaçant les joueurs quand le jeu n'est pas en mode multijoueur.

Par conséquent, les ennemis doivent pouvoir se déplacer sur la carte mais aussi attaquer les joueurs. Quant aux personnages principaux non joueurs, ils doivent pouvoir suivre le personnage principal jouable et se comporter comme un joueur dans un combat (cette partie sera expliquée plus en détail dans la suite du rapport).

Lors de la première soutenance, seul un début de l'implémentation des mouvements des IA était réalisé. En effet, les IA peuvent bouger sur le terrain et suivre le joueur. A cette étape, elles ne pouvaient pas attaquer et les animations n'étaient pas présentes. De surcroît, ce système n'était opérationnel que pour les IA remplaçant les joueurs et ne l'était pas pour les ennemis et PNJ qui utilisent un autre système de déplacement (cette partie sera plus détaillée par la suite).

II. Présentation générale du projet et des tâches individuelles

Avancement général du projet

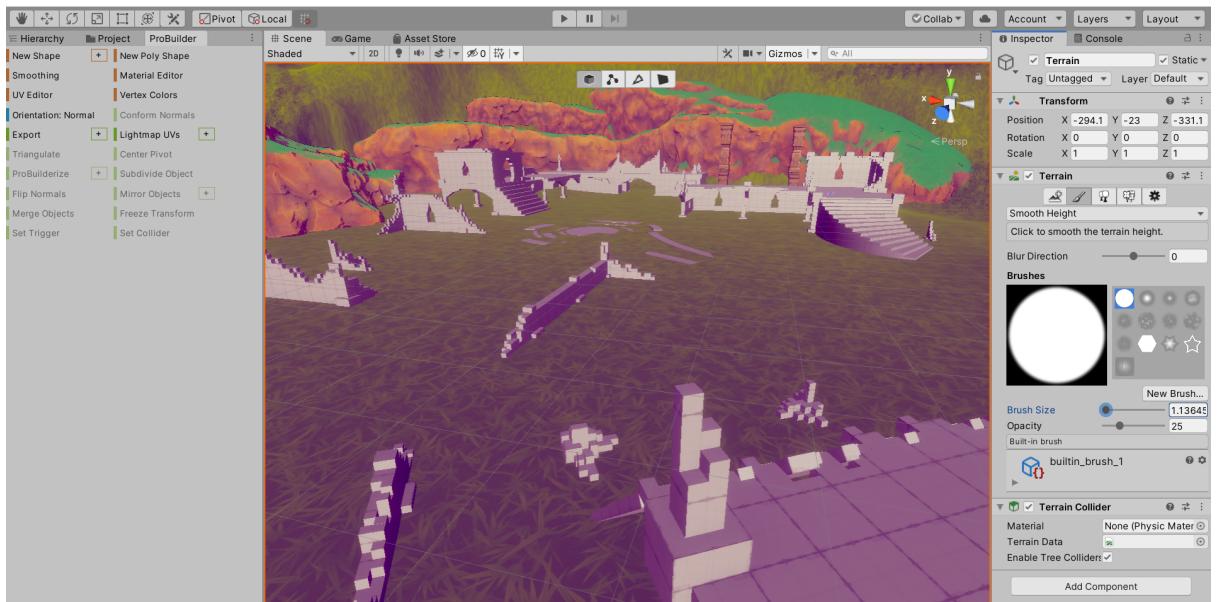
- Présentation du planning de réalisation

Nous avons mis à jour le tableau de l'avancé des tâches, celui-ci a varié par rapport à celui du cahier des charges et du dernier rapport.

Tâches à Faire	Déjà fait	Avancement par Soutenance (en %)		
		Soutenance 1	Soutenance 2 (Stade Actuel)	Soutenance 3
Models 3D		40	100	100
Cinématiques		0	0	100
Level Design		40	90	100
Menus		95	95	100
Dialogues		30	40	100
Musiques		50	100	100
Placer les musiques		0	80	100
Placer les sons FX		5	70	100
Gestion des Lumières		75	95	100
PlayerController		60	100	100
Animations		30	95	100
Système de Sauvegarde		100	100	100
Système Multijoueur		100	100	100
Ennemis et PNJ		40	90	100
Système de sous-titres et sélection de langues		100	100	100
Système de Volume		100	100	100
Système de combat		30	90	100
Système de communication		100	100	100

- Level design
 - **Constructions**

Pour pouvoir créer les “scènes” (map) sur lesquelles nous allons jouer, nous avons utilisé Pro Builder, un outil déjà intégré dans Unity. Facile d'utilisation, il demande cependant un certain entraînement et surtout beaucoup, beaucoup de temps pour obtenir un résultat intéressant.



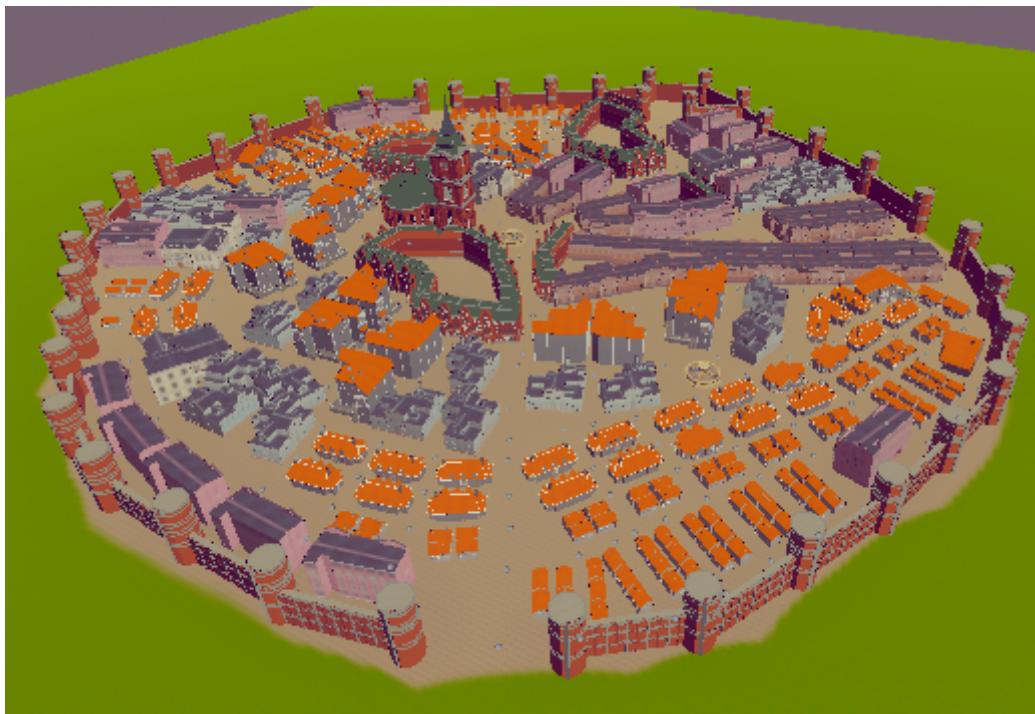
La jungle a été faite par Léo ALBA.

Quant au Temple Aquatique (intérieur), il a été fait par Léo FARHI :



Concernant la construction des villes, nous avons d'abord utilisé un algorithme pour placer des cubes indiquant où il faut poser une maison et comment l'orienter afin de créer une ville. Suite à cela, Hugo a remplacé tous ces cubes par des "Assets" de bâtiments pour le rendu final. Nous avons procédé de cette manière pour rendre la tâche de la création de la ville plus simple et plus rapide. En effet, cela aurait pris plus de temps pour réfléchir au design de la ville puis de la construire.

Après la création de la ville, Hugo a placé un par un tous les nœuds constituant le "Population Engine" (cette notion est expliquée dans la partie IA du rapport), puis a relié tous les nœuds entre eux afin de pouvoir implémenter les PNJ.



- Générateur de PNJ avec tenue semi-aléatoire

Le système de générateur de PNJ permet de créer des PNJ qui ont des tenues et des parties du corps différentes afin de rendre une ville réaliste. Ce système a été fait par Léo FARHI et les modèles 3D quant à eux, ont été fait par Geoffrey ELBAZ à l'aide de Vroid et de Blender.

Le principe de ce système est relativement simple contrairement à sa réalisation. On fournit au système une multitude de modèles 3D (vêtements, cheveux, pantalons, yeux, etc...) et le système sélectionne de façon aléatoire ces éléments et crée une combinaison qui forme à la fin un PNJ. Mais le système n'est pas bête, il vérifie si les éléments (exemple : vêtement) sont en adéquation avec le genre du PNJ, autrement dit, il ne va pas mettre des tenues de femme sur un homme et vice-versa.



Les informations liées aux éléments comme par exemple le genre, l'emplacement du model 3D, etc... sont sauvegardés dans un “ScriptableObject” et oui, encore eux !

Si vous vous souvenez bien, nous en avons parlé dans le dernier rapport pour expliquer le système de sauvegarde. Mais un petit rappel ne fait pas de mal.

Standar Girl

Open

Script	CharacterCreatorMesh
Name Part	
Sex	Girl
Type	Body
Model 3D	Standar Girl
Mesh Name	Body

GirlJacket1

Open

Script	CharacterCreatorMesh
Name Part	
Sex	Girl
Type	Cloth
Model 3D	vetement
Mesh Name	Body

En résumé, un ScriptableObject est un type de Class spécifique à Unity. Ce type permet de concrétiser une variable en permettant de la stocker dans un fichier. Ainsi, sauvegarder les différentes parties d'un modèle 3D n'a jamais été aussi simple.

Bien que les ScriptableObject nous aient facilité la tâche, il y a eu tout de même une partie très complexe : synchroniser les vêtements avec le squelette du PNJ.

Pour ce faire, Léo FARHI a créé un système qui permet de fusionner les deux squelettes (celui du vêtement et celui du joueur).

Tout d'abord il fallait vérifier si les deux squelettes avaient les mêmes os “Bones”, donc l'algorithme parcourrait de manière récursive à travers les deux squelettes pour s'assurer qu'ils avaient bien des parties identiques. Si ce n'est pas le cas alors le système recrée les parties manquantes.

```
⌚ Frequently called 2 usages Leo Farhi
private void ChildInstantiate(GameObject obj, GameObject TargetRoot)
{
    if (null == obj)
        return;
    foreach (Transform child in obj.transform)
    {
        if (null == child)
            continue;
        Transform childTarget = TargetRoot.transform.Find(child.gameObject.name);
        if (childTarget == null)
        {
            GameObject Instance = new GameObject(); //Instantiate(new GameObject(), new Vector3(0, 0, 0), Quaternion.identity);
            Instance.name = child.gameObject.name;
            /*
            Debug.Log(child.gameObject.name);
            Debug.Log(child.parent.gameObject.name); */
            Instance.transform.parent = TargetRoot.transform;
            Instance.transform.localPosition = child.localPosition;
        }
        ChildInstantiate( obj: child.gameObject, TargetRoot: TargetRoot.transform.Find(child.gameObject.name).gameObject);
    }
}
```

Puis dans un second temps, le système fusionne les deux squelettes pour n'en former qu'un.

```

public void ReBone()
{
    ChildInstantiate( obj: root, rootTarget);

    Dictionary<string, Transform> boneMap = new Dictionary<string, Transform>();
    foreach (Transform bone in TargetMeshRenderer.bones)
        boneMap[bone.gameObject.name] = bone;

    SkinnedMeshRenderer myRenderer = OriginMeshRenderer;//gameObject.GetComponent<SkinnedMeshRenderer>()
    myRenderer.rootBone = rootTarget.transform;

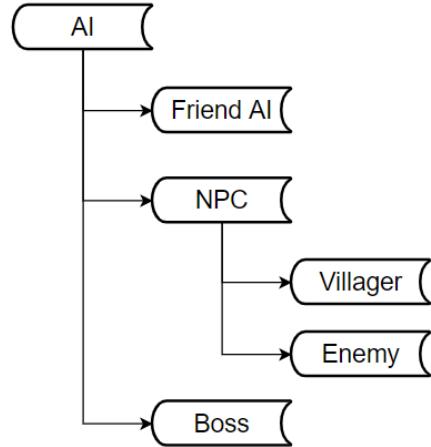
    Transform[] newBones = new Transform[myRenderer.bones.Length];
    for (int i = 0; i < myRenderer.bones.Length; ++i)
    {
        GameObject bone = myRenderer.bones[i].gameObject;
        if (!boneMap.TryGetValue(bone.name, out newBones[i]))
        {
            newBones[i] = GetChildRecursive(rootTarget, bone.name).transform;
            /*
            Debug.Log("Unable to map bone \"" + bone.name + "\" to target skeleton.");
            break;*/
        }
    }
    myRenderer.bones = newBones;
}

```

Mais ces opérations sont très coûteuses à calculer pour la machine.

- Système d'intelligence artificielle

En utilisant la programmation orientée objet (POO) et le langage de programmation C#, l'architecture des scripts des IA est faite comme suit :



Architecture des scripts pour les IA

La classe mère "AI" regroupe tous les attributs et les méthodes qui seront utilisés dans les différentes classes filles : "Friend AI", "NPC" et "Boss". En particulier, c'est dans cette classe que seront écrites les méthodes concernant la mort, l'application des dommages, et les différentes fonctions pour attaquer. En effet, cela est pratique de pouvoir utiliser les attaques d'un boss pour un ennemi juste en changeant l'animation qui est à part.

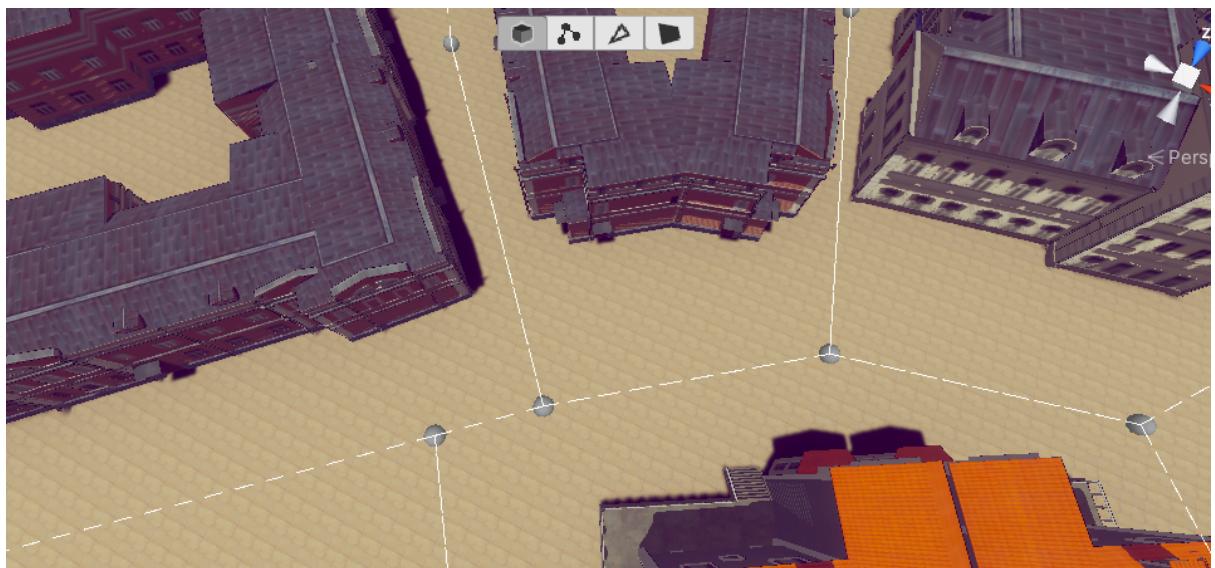
Pour les classes filles "Friend AI" et "Boss", elles regroupent plus précisément le mouvement des IA (expliqué plus bas) et les différents patterns (ici on parle d'un enchaînement de plusieurs attaques par exemple) propre à ces classes. La classe "Boss" aura autant de sous-classes qu'il y aura de boss. Cela permet donc d'avoir plusieurs boss avec des patterns différents selon chacun.

Enfin la classe fille "NPC" contient deux sous-classes "Villager" et "Enemy". En fait, ces scripts ont été placés de cette manière car les PNJ villageois et les ennemis ont le même déplacement. Les avoir séparés nous permet de mieux nous y retrouver. Ainsi, la classe "NPC" contient le déplacement des PNJ et les sous-classes "Villager" et "Enemy" contiennent respectivement tout ce qui est lié aux PNJ (à savoir interaction avec le joueur) et tout ce qui est lié aux ennemis (à savoir le pattern des combats).

o Les déplacements

Tous les déplacements des IA sont gérés en utilisant les propriétés du “Navmesh” qui est un système intégré à unity qui sert à faire du pathfinding, un autre système permettant à un objet de trouver le chemin le plus court pour atteindre une destination. Les IA peuvent donc à chaque instant trouver la position du joueur et se diriger vers lui en cherchant le chemin le plus court. Le Navmesh permet donc une implémentation simple et efficace des IA qui peuvent se déplacer vers une cible.

Le “Population Engine” a été fait par Léo FARHI et est utilisé par Hugo pour implémenter les PNJ. C'est un système de guidage pour les IA qui se baladent dans la ville. Ce système fonctionne à l'aide de nœuds (Node) et de lignes (Edge), les nœuds représentent chaque intersection d'une ville quant aux lignes, elles représentent les routes que peuvent emprunter les PNJ.



Lorsqu'un PNJ arrive sur un nœud, celui-ci est redirigé vers un autre. Pour faciliter l'installation de ce système sur les villes, Léo F. a dû modifier le comportement de l'éditeur afin qu'on puisse voir les lignes et les nœuds dans l'éditeur et non en jeu à l'aide des “Gizmos”.

Par conséquent, les nœuds sont les cibles des PNJ utilisant le Navmesh. Pour se déplacer, les PNJ vont chercher le chemin le plus court pour arriver au prochain nœud et avancer vers lui.

Comme dit dans les rappels, les IA qui remplacent les joueurs doivent suivre le joueur principal. Ils vont donc utiliser le Navmesh et avoir comme cible le joueur. Cela aura pour conséquence un déplacement des IA vers le joueur à chaque fois que le joueur se déplace. Ce système de déplacement a été pensé et implémenté par Hugo.

Par conséquent, ce déplacement fonctionne de la façon suivante : si l'IA est à une distance proche du joueur alors elle marche. Si elle est à mi-distance alors elle court et enfin si elle est loin du joueur alors elle sprinte. Le déplacement est donc facilement implémenté par une suite de conditions comme le montre le code suivant :

```
5 références
public override void AIMove()
{
    distanceTarget = Vector3.Distance(target.position, transform.position);

    //A améliorer
    if (distanceTarget > distanceIdle)
    {
        if (distanceTarget > distanceSprint)
        {
            ChangeAnimation("sprint", true, sprint);
        }
        else if (distanceTarget > distanceRun)
        {
            ChangeAnimation("run", true, speedRun);
            ChangeAnimation("sprint", false, speedRun);
        }
        else if (distanceTarget < (distanceRun - 0.1))
        {
            ChangeAnimation("run", false, speedWalk);
            ChangeAnimation("walk", true, speedWalk);
        }

        agent.destination = target.position;
    }
    else
    {
        ChangeAnimation("walk", false, 0);
    }
}
```

La méthode *Distance* permet de calculer la distance entre le joueur et l'IA, *ChangeAnimation* permet de changer d'animation (détails dans la partie animation) et de changer la vitesse de déplacement de l'IA (en effet, si elle court, elle sera alors plus rapide que si elle marche !), et enfin *agent.destination* permet d'utiliser le Navmesh et faire en sorte que l'IA se déplace vers le joueur en prenant le chemin le plus court.

○ Les combats

Concernant les combats, tout le système qui suit a été principalement pensé et fait par Hugo avec l'accord des autres membres du groupe. Les ennemis standards n'auront que la possibilité d'attaquer ou de se déplacer pendant le combat de façon aléatoire. Les boss quant à eux auront leurs propres scripts donc leurs propres styles de combat. Bien évidemment, tous les boss attaquent et se défendent. Enfin, pour les IA qui remplacent les joueurs, leur système de combat est plus complet que celui des ennemis standards.

Dans le jeu, un combat se déclenche lorsque le joueur ou l'IA est assez proche d'un ou plusieurs ennemis. Il a été décidé que c'est l'ennemi qui déclenche un combat pour éviter tout bug avec les autres IA (sauf si le joueur attaque en premier). Un combat se termine si tous les ennemis sont morts ou si tous les joueurs meurent.

Ce système est géré suivant un “diagramme à conditions” (que vous pouvez voir plus bas dans cette partie). Pendant les combats, les IA ont différents états mais ne peuvent pas en avoir plusieurs en même temps. Ces états sont : Dodge (esquive), Impact, Random Move (mouvement aléatoire), Attack (attaque), Dead (mort) et enfin None (nul) qui est l'état par défaut. Ces différents états sont activés en fonction de plusieurs conditions. Par exemple, l'état Attack ne peut s'activer que si l'IA est à une distance proche de l'ennemi. Ainsi, si aucune condition n'est remplie alors l'IA va se déplacer de façon aléatoire. En fait, pendant un combat, l'IA se déplacera de façon aléatoire (avec une probabilité plus grande de se rapprocher de son ennemi) jusqu'à ce qu'une condition soit remplie et change son état.

Par la suite, une fois que l'IA a atteint un état (Dodge, Impact ou Attack), c'est là qu'un système de probabilité intervient. En effet, comme il s'agit d'une sorte d'IA, on ne veut pas qu'elle exécute une action dès qu'une condition est remplie (un humain ne ferait jamais cela). C'est pour cela que l'on introduit ce système de probabilité. L'IA aura donc une certaine probabilité de faire son action.

Par ailleurs, cette probabilité est variable en fonction de l'ennemi. Par exemple, un ennemi faible aura une probabilité plus faible d'attaquer et une probabilité nulle de d'esquiver. Alors qu'un ennemi plus fort pourra esquiver et attaquer plus fréquemment.

Enfin, certains états sont prioritaires sur les autres (ici “Impact” et “Dead”). Cela signifie que si la condition pour activer cet état est vérifiée alors que l'IA est dans un autre état, c'est l'état prioritaire qui sera actif. Cela permet par exemple d'éviter que l'IA bouge alors qu'elle est morte ou que l'IA continue d'attaquer alors qu'elle vient de se prendre un coup (si la probabilité de l'état “Impact” est vérifiée bien sûr).

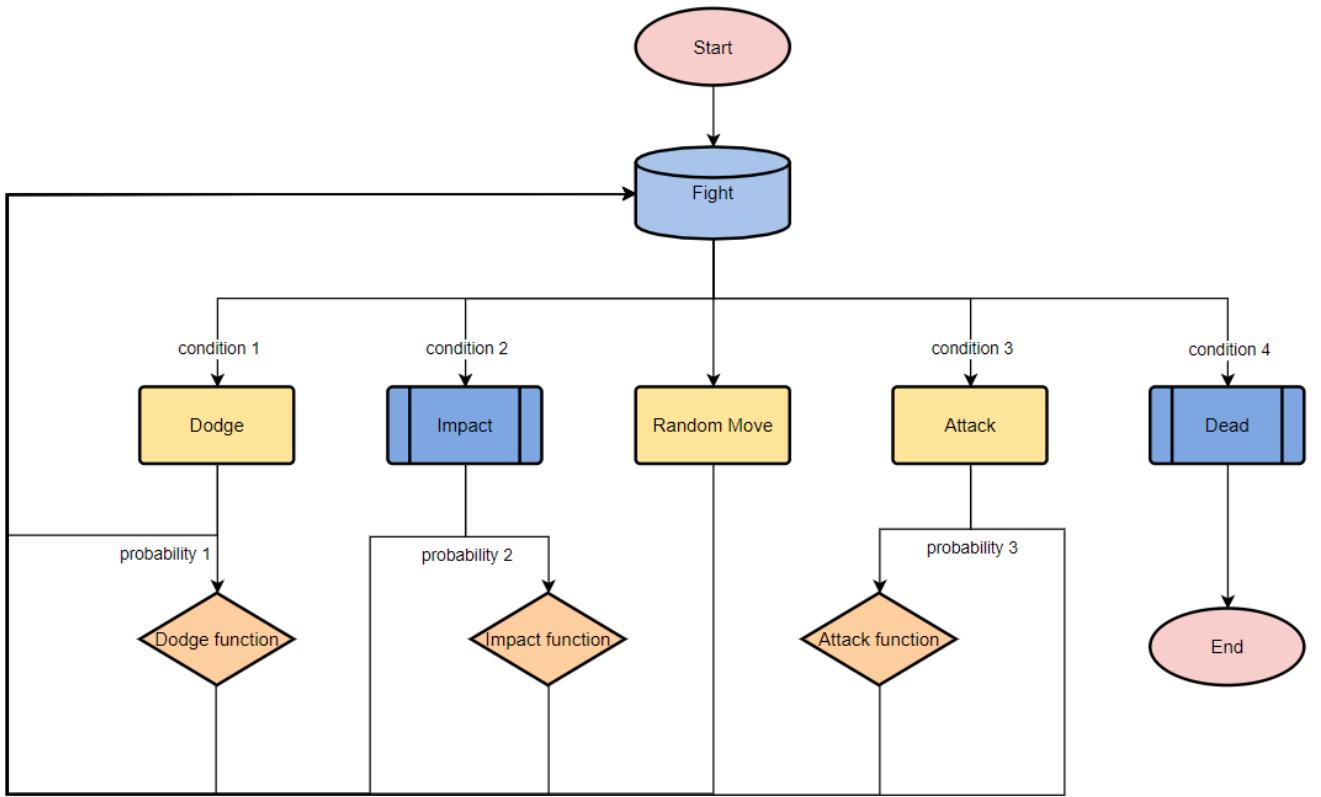


Diagramme de fonctionnement des combats des IA

L'idée est là mais l'implémentation doit encore être améliorée. Il reste pas mal de bugs quand deux IA combattent même si le système fonctionne assez bien quand une IA combat un joueur.

Par ailleurs, ce diagramme n'est pas complet. En effet, on peut ajouter autant d'état que l'on veut une fois que le système est bien opérationnel comme par exemple un état "block" qui permet à l'IA de parer ou de bloquer une attaque adverse.

- Les animations

Pour les animations, comme nous l'avons précisé dans le cahier des charges et dans la rapport de la première soutenance, nous avons importé des animations. Ensuite, Hugo et Léo FARHI ont utilisé ces animations dans l'Animator de Unity pour les appliquer aux personnages du jeu.

Une fois les animations mis dans l'Animator, celles-ci sont contrôlées dans le script des différents personnages (joueur ou IA). De surcroît, il y a autant d'Animator qu'il y a de personnages ayant des animations différentes. La grande difficulté des animations est de créer un lien entre elles pour rendre le visuel fluide. En effet, une mauvaise gestion de l'Animator et des propriétés des animations peut avoir pour conséquence des bugs visuels. Il est donc important d'avoir une bonne gestion des animations.

- Musiques

Geoffrey ELBAZ s'est chargé de composer 5 musiques pour les différents lieux ou moments du jeu:

- le menu principal
- l'intérieur de la prison
- l'intro de la jungle
- la jungle
- la ville n°1

Quant aux autres musiques, ce sont des intelligences artificielles comme AIVA qui les ont composées, et pour d'autres nous avons pris des musiques libres de droit.

- Network

Comme dit dans le dernier rapport, le Network (la partie multijoueur) était fini à 100%. Sauf que nous avons oublié quelques petits détails notamment la synchronisation de tous les PNJ et Items se trouvant sur la Map chez tous les clients. Chose qui a donc été faite par Léo FARHI et il a créé un système qui s'adapte à toute situation grâce au ScriptableObject et, par conséquent, si nous oubliions encore certaines choses en rapport avec le Network il sera très facile de les implémenter.

Par exemple, ces deux fonctions permettent de synchroniser l'apparence des PNJ chez tous les joueurs :

```
private void SerializeData()
{
    this.m_StreamQueue.SendNext(DataPNJ);
}

private void DeserializeData()
{
    string data = "";
    try
    {
        data = (string) this.m_StreamQueue.ReceiveNext();
        if (data=="")
            return;
        GameObject.FindObjectOfType<PNJgenerator>().GenerateWithBase(this.gameObject, data);
        Destroy(this);
    }
    catch{;}
}
```

Celui qui héberge la session envoie à l'aide de `SerializeData` les informations relatives aux PNJ, et les clients reçoivent les données à l'aide de `DeserializeData` qui récupère l'information, puis la transmet au `PNJgenerator` afin que le système puisse générer un PNJ identique à celui de l'hébergeur.

Le système de communication est composé de deux principaux types d'envoi de données. Le premier principe à pour but de synchroniser la même variable pour tout le monde (par exemple, l'heure du système jour-nuit). Quant à l'autre type de données, c'est la conservation indépendante des données des joueurs (par exemple, le choix de notre joueur, chaque client envoie le personnage qu'il a choisi au serveur sans pour autant modifier la valeur des autres clients, cette donnée est purement indicative).

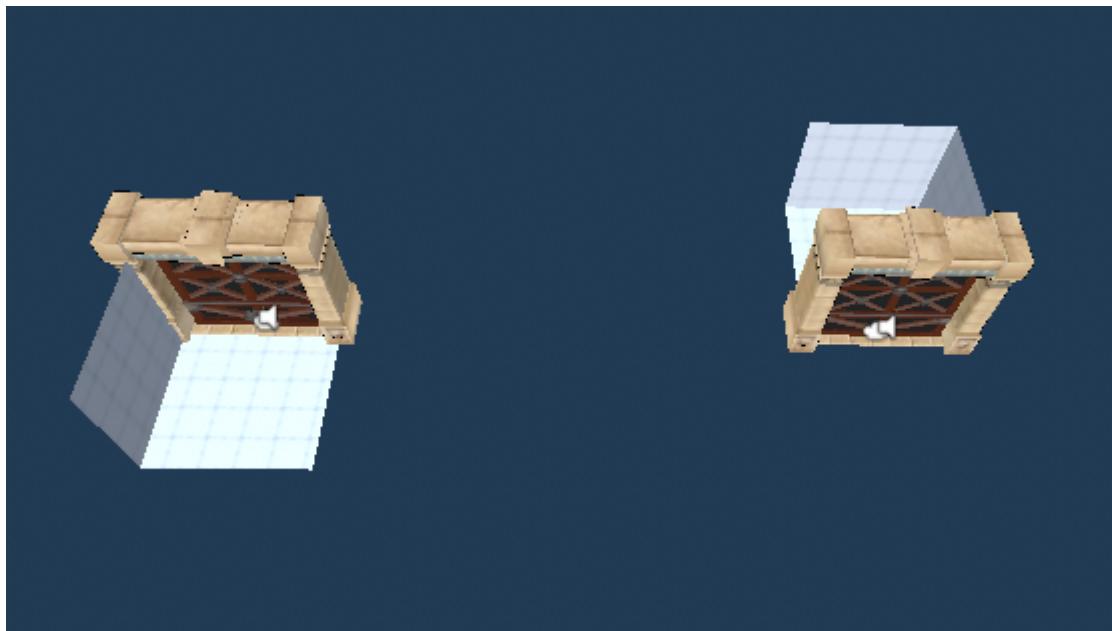
- Optimisation

La partie optimisation est essentielle afin qu'un ordinateur peu puissant puisse faire tourner le jeu, mais il est préférable d'avoir une carte graphique et un bon processeur pour être sûr de n'avoir aucun souci.

Pour optimiser le jeu du point de vue visuel, nous utilisons l'occlusion culling. Ce système implémenté par Unity permet de cacher les éléments qui ne sont pas dans le champ de vision de la caméra et donc d'alléger grandement le rendu graphique.

Nous utilisons aussi un système de téléportation fait par Léo FARHI, ce système a deux fonctions :

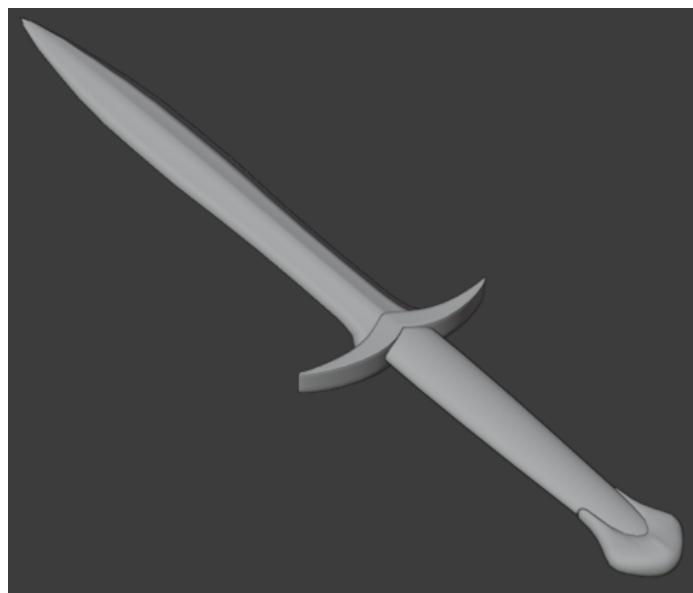
- Faciliter les échanges entre les différents lieux
- Alléger le rendu graphique



En pratique, le joueur rentre d'un côté et en sort de l'autre. Quand le joueur s'éloigne des portes, un écran noir se place devant la porte et le système désactive le portail afin d'alléger le rendu graphique.

- Modèles 3D

Suite à une décision générale du groupe concernant la modélisation 3D, nous avons dû revoir notre approche dans cette partie. En effet, désormais nous allons aussi utiliser des Assets afin de nous faciliter le travail et de gagner un temps précieux. Pour autant, nous n'avons pas abandonné l'idée de réaliser nos propres modèles 3D. Nous nous concentrerons plus sur la modélisation des objets uniques de notre jeu comme les armes. Pour cela Geoffrey a modélisé des objets tels que certaines armes pour les personnages principaux ou les ennemis à l'aide du logiciel Blender.



Pour les Personnages Non Joueurs (PNJ) nous avons choisi de faire chaque partie du corps indépendamment afin de pouvoir faire beaucoup plus de combinaisons. Pour les femmes nous avons fait 3 coupes de cheveux, 3 hauts, 3 bas. Nous avons fait exactement la même chose pour les hommes ainsi que 2 perruques, 2 hauts et 2 bas, ce qui fait 125 possibilités d'habits pour les femmes et les hommes. Comme vous avez pu le deviner, le générateur de PNJ expliqué précédemment pourra générer jusqu'à 125 PNJ différents ce qui est amplement suffisant pour donner de la vie dans les villes.

- Déduction

Comme vous avez pu le voir plus haut, notre groupe se base sur un tableau d'organisation, pour savoir ce qui doit ou non être fait et pour nous rappeler quelles sont les priorités. Bien que nous ayons pris du retard suite à certains événements comme les mid-terms ou encore certains problèmes personnels qui nous ont empêché d'avancer sur le projet efficacement, nous avons réussi à rattraper ce retard durant la semaine de vacances.

Avancement général du site Web

Ce site web est le même que celui présenté lors de la première soutenance dans la forme, bien entendu le fond a été modifié pour correspondre à nos attentes artistiques et techniques. Les responsables du site web sont les mêmes que lors de la première soutenance, soit Geoffrey ELBAZ pour le code et Léo FARHI pour le design. La page d'accueil comporte les liens pour les différentes pages demandées sous forme d'en-tête, ce qui permet une présentation propre et efficace. Elle contient en plus un lien très visible vers la page de téléchargement pour inciter les utilisateurs à télécharger le jeu. Chaque page a la même présentation globale pour permettre une cohérence artistique et une utilisation plus intuitive.

Problèmes rencontrés

- Réalisation

Un certain manque d'efficacité a pu être remarqué chez certains et des erreurs de manipulation ont mené à la perte d'un assez lourd travail de l'un des créateurs (qui ne postait pas régulièrement son travail sur Github).

Au début du projet, nous avions l'intention de tout faire par nous-même. Cela prend en compte les modèles 3D des personnages et des bâtiments. Mais, comme nous l'avons dit un peu plus tôt dans la partie des modèles 3D, suite à un manque crucial de temps (à cause des cours à côté principalement), nous avons abandonné l'idée de créer tout nous-même, et d'importer des assets ce qui nous a fait gagner beaucoup de temps, notamment pour les bâtiments et les objets de décoration comme des fontaines ou des bancs pour les villes. Mais cette décision a aussi eu comme conséquence de rendre le

travail de certains d'entre nous inutile. En effet, certains membres du groupe ont commencé à modéliser des bâtiments incomplets pour le moment. Ils avaient passé un temps non négligeable pour faire cela et leur travail a été réduit à néant pour le bien de tous et la réussite du projet.

- Entente du groupe

En général, comme nous sommes dans la même classe, il y a une bonne ambiance dans le groupe, mais à noter que le chef du projet met parfois un peu la pression afin de respecter au plus proche le tableau d'avancement.

- Bugs et solutions

Comme il fut évoqué précédemment, un des créateurs a perdu une bonne partie de son travail car il ne le postait pas régulièrement. En effet, ce qu'il avait créé a fini par être corrompu. La solution fut alors pour lui de charger une backup ce qui lui a fait malheureusement perdre plusieurs heures de travail.

Il y a aussi un bug avec le système de portail qui affiche mal l'eau quand on regarde à travers celui-ci, c'est dû à un problème de rendu graphique.

Les bugs d'animation ont été certainement les bugs les plus amusants à voir (comme lorsqu'une IA attaque, elle glisse en même temps sur le sol ou encore des roulades dans le vide etc...). Mais ces derniers ont été aussi les plus difficiles à régler. En effet, comme nous ne sommes pas très habitués aux systèmes et propriétés des animations, trouver des solutions à ces problèmes nous a pris beaucoup de temps et de ressources pour un résultat qui n'est pas forcément de haute qualité.

III. Conclusion

Rappel sur ce qui est fait

- Modèles 3D

En effet, tous les modèles 3D qui servent à composer le jeu ont été faits, soit par Vroid, soit par Blender, soit par Pro Builder.

- Network

Cette partie là est finie à 100% bien qu'il y ait quelques bugs mineurs à résoudre.

- Système de sauvegarde

A été fait depuis le tout début du projet et fonctionne à l'aide des ScriptableObject.

- Musiques

- Système de sous-titre/dialogue et sélection de langue

Rappel sur ce qui est à faire

- Finir le site web à 100%
- Faire les dialogues des PNJ
- Player Controller

La partie du mouvement du player controller était déjà fonctionnelle depuis la première soutenance, néanmoins ce n'est pas le cas de la partie combat qui est à finir. Cette partie n'est pas la plus compliquée puisque nous allons réutiliser les méthodes des IA pour le joueur.

- Finir le level design

Dû à notre retard évoqué plus haut nous n'avons pas pu terminer à 100% le level design. En effet la Scène "jungle" de notre jeu est une partie importante qui est toujours en cours de construction/de finalisation. Pour éviter que cette zone soit juste une route sans intérêt particulier, nous avons décidé de faire un chemin d'obstacles et de puzzle/énigmes qui vous attendent. Cependant son arrivée ne devrait pas tarder car une très grosse partie à déjà été faite.

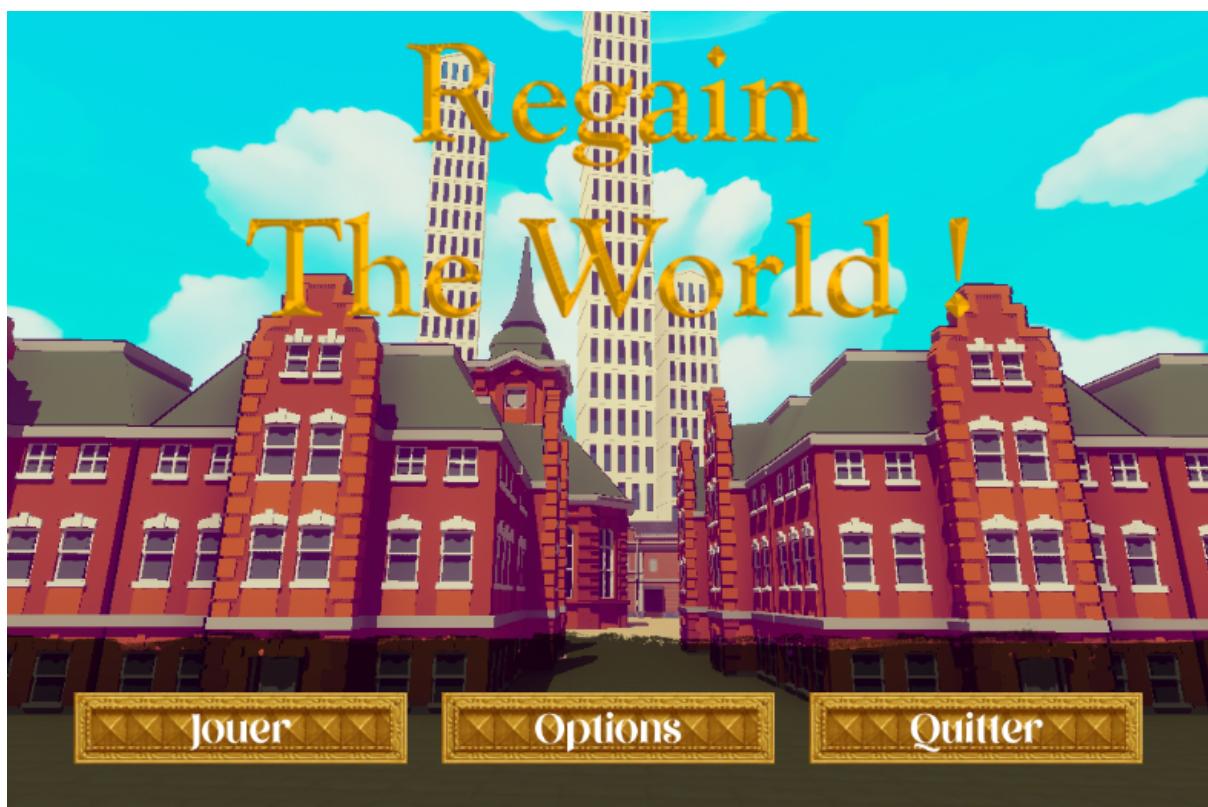
- IA

Les principales fonctionnalités de l'IA sont quasiment opérationnelles. Il faut désormais rendre le système de combat plus fluide en supprimant les éventuels bugs visuels lors des combats. Bien évidemment, il reste l'implémentation des boss à réaliser et l'ajout si nécessaire de nouvelles fonctionnalités lors des combats.

- Faire les cinématiques

IV. Annexes

Images du jeu



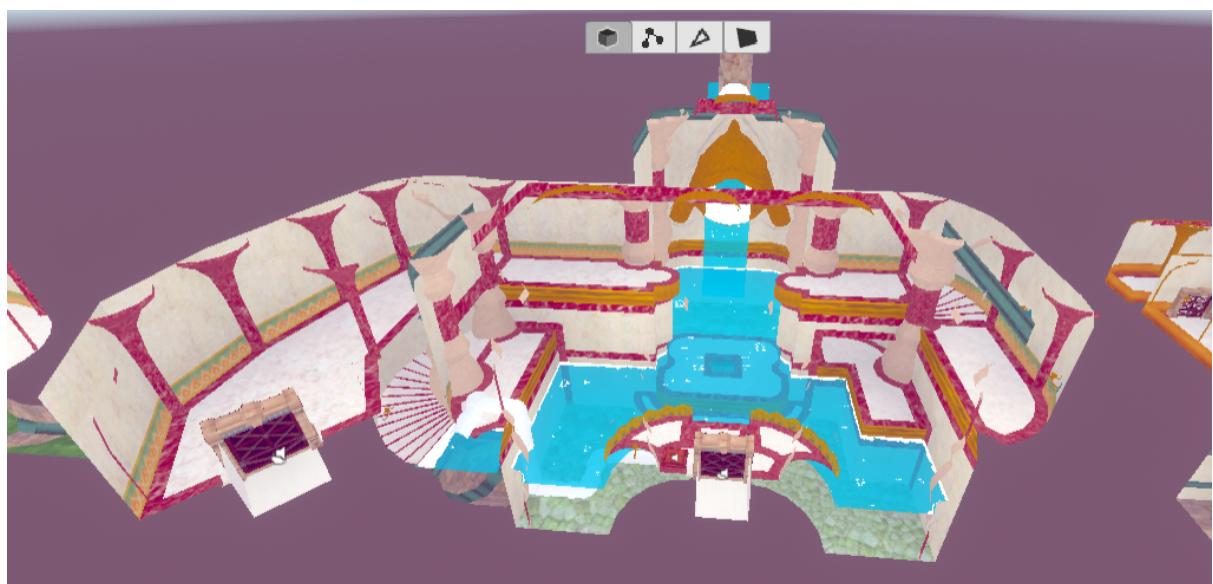
Petite astuces pour alléger le menu principal :

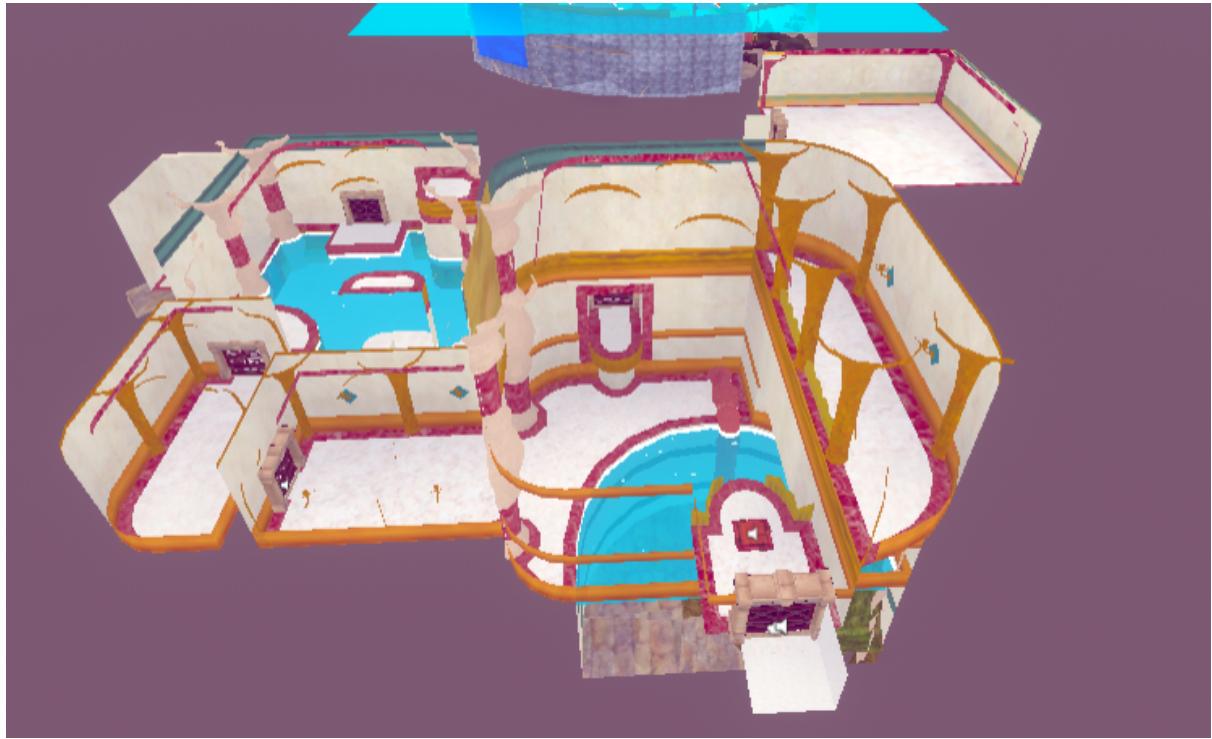


Temple se trouvant dans la Jungle fait par Léo FARHI (Inspiration : Zelda Skyward Sword) :



Salles vu de l'extérieur :





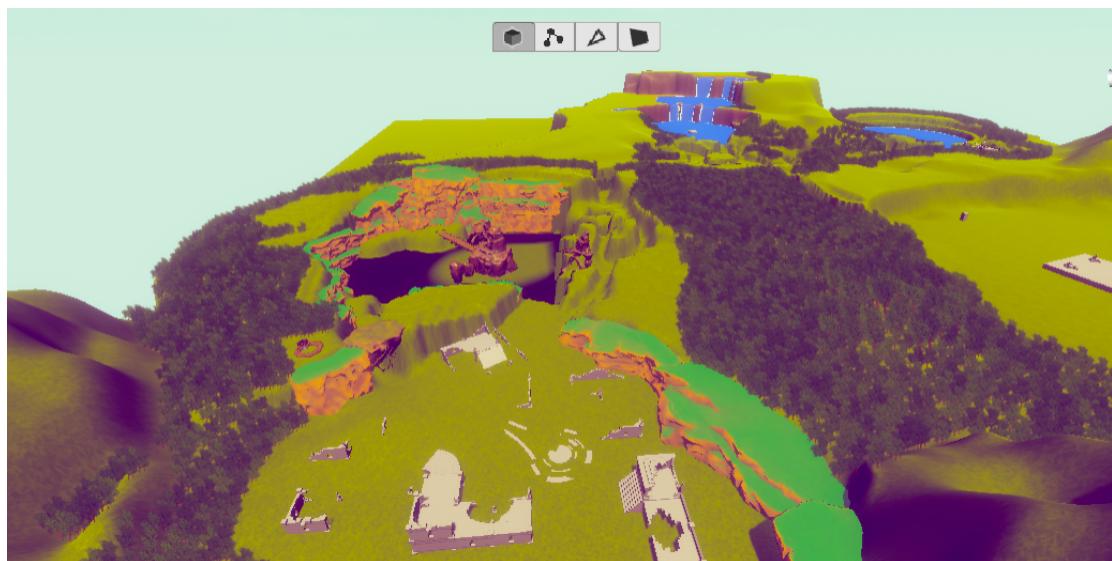
Ville n°1 faite par Hugo RUBIO, (les traits blancs font partie du “Population Engine”, ils représentent les routes que peuvent emprunter les PNJ) :



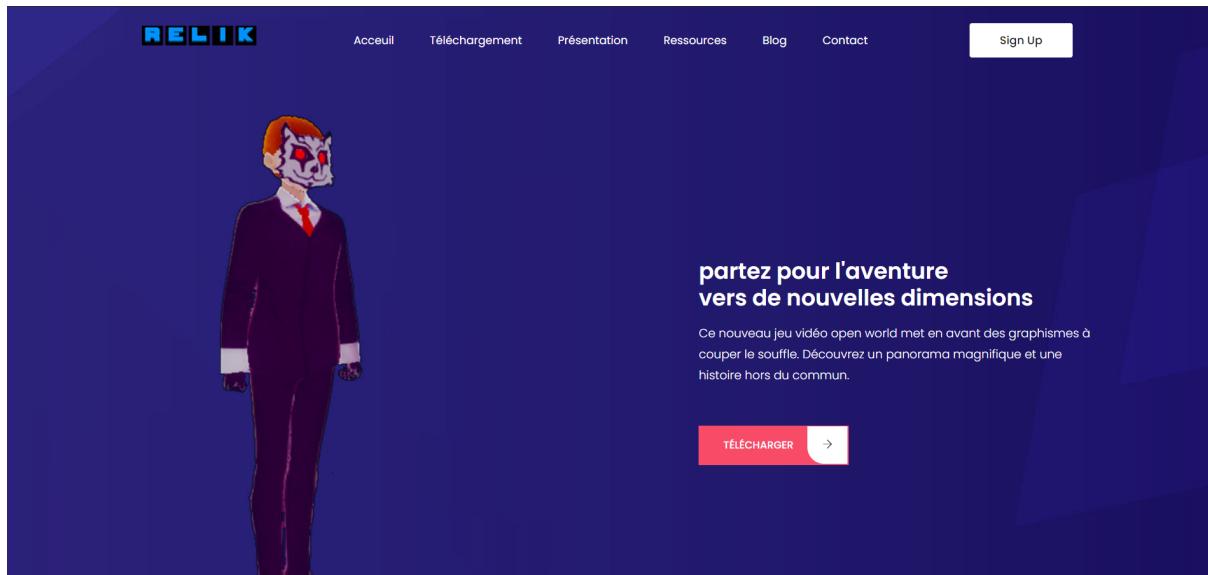
Ville n°2 :



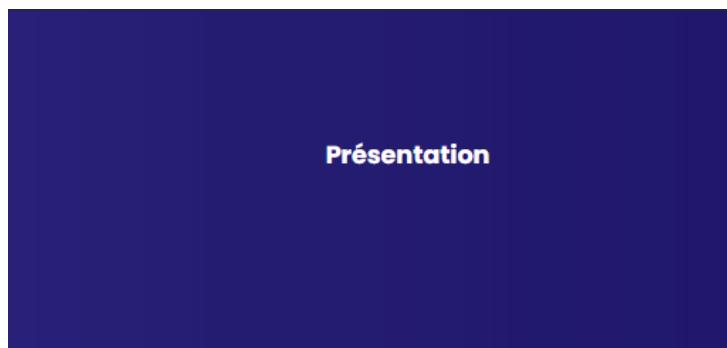
Jungle fait par Léo ALBA (Ruines, objets divers et terrain) et Léo FARHI (Cascade et terrain) :



Site Web



Affichage un peu spécial car elle a été dézoomée pour être prise dans son intégralité :



Équipe

Ici vous trouverez chaque membre du projet.

A vertical list of four team members, each with a small profile picture and their name below it. The profiles are contained within light gray boxes.

- Léo Farhi
- Geoffrey Elbaz
- Léo Alba
- Hugo Rubio

Téléchargement

[TÉLÉCHARGER LE JEU](#)[TÉLÉCHARGER LA VERSION LITE](#)[TÉLÉCHARGER LE RAPPORT](#)