

Rapport Final

Nom de Projet : *Pricefield Garden*

Nom de groupe : Arcadia

Belkhder Ilias

Martinez Alexandre

Poelger Jonathan

Brice Benoît

Mai 2020

Encadrants : Remi Vernay, Philippe Roussille



Table des matières

1	Introduction	2
1.1	Analyse pré-dévelopement	3
1.1.1	Origine	3
1.1.2	Objet d'étude	3
1.1.3	État de l'art	4
2	Découpage du projet	5
2.1	Avancement des tâches	5
3	Avancement	6
3.1	Interface	6
3.1.1	Menu principal (Alexandre M.)	6
3.1.2	Menu in-game (Ilias B. et Jonathan P.)	10
3.1.3	Ecran de chargement (Ilias B. et Alexandre M.)	11
3.2	Map	12
3.2.1	Modélisation (Benoit B.)	12
3.2.2	Objets et textures (Jonathan P. et Benoit B.)	22
3.2.3	Examinables (Jonathan P.)	23
3.2.4	Collisions (Jonathan P.)	23
3.2.5	Environnement (Jonathan P.)	24
3.3	Personnage (Jonathan P.)	25
3.3.1	Modèle	25
3.3.2	Animations	26
3.3.3	Caméra	26
3.3.4	Esprit	27
3.3.5	Déplacements	27
3.3.6	Cinématiques	28
3.3.7	Téléportation	28
3.3.8	HUD	29
3.4	Sauvegarde (Alexandre M.)	29
3.5	Enigmes (Ilias B.)	31
4	Site web (Ilias B. et Jonathan P.)	34
5	Récit de la réalisation	35
6	Conclusion	38
7	Annexes	39

1 Introduction

Après un semestre entier de travail de groupe, le projet Pricefield Garden touche enfin à sa fin. Malgré les difficultés rencontrées, les nombreux bugs et une première partie de développement compliquée, beaucoup de travail a été effectué pour pouvoir atteindre ce stade final de notre jeu vidéo. Malgré que nous soyons pour le moment fier de là où nous en sommes rendus, il reste quelques bugs à résoudre tels que les temps de chargement un peu trop longs, les énigmes et les téléporteurs qui se mélangent du fait des noeuds Godot qui reçoivent des signaux qui ne leur sont pas dus.

Le projet Pricefield Garden a été développé en C# et en utilisant le moteur de jeu Godot. N'étant initialement pas familier avec ce moteur de jeu, il nous a fallu un certain temps d'adaptation afin de comprendre ses mécaniques et ses spécificités.

Le jeu est disponible à l'adresse suivante¹ et contient un endroit où télécharger le jeu, la soluce complète, le profil des membres de l'équipe et une brève présentation du jeu.

Pour rappel, Pricefield Garden est ainsi un jeu solo en vue à la troisième personne, où le joueur incarne un personnage qui se réveille dans une plaine aux côtés d'une lupiotte. Rapidement, il se rend compte qu'il est bloqué sur une île, entourée d'une part d'une mer qui n'en finit pas, et de l'autre d'un immense mur. Il comprend que pour s'échapper, il lui faudra résoudre les différentes énigmes qui arrivent sur son chemin et trouver la sortie. Ici, le joueur devra explorer la carte et les environnements mis à sa disposition, tout en résolvant les énigmes qui lui seront proposées afin d'avancer dans cette quête. Il va devoir s'aider de la lupiotte afin de franchir les différents niveaux qui se succèdent. Chaque région de la carte consiste en une succession d'énigmes. Il devra, par ailleurs, trouver une explication à ces épreuves.

1. <https://arcadios-cr.github.io/Main.html>

1.1 Analyse pré-dévelopement

1.1.1 Origine

Il s'agit d'un jeu vidéo d'exploration basé sur un personnage que l'on va maîtriser et suivre tout au long de l'aventure. L'idée de ce jeu nous est venue en parlant du projet entre nous ; l'un de nous a proposé de faire un jeu d'exploration, tout en citant "Journey", ce qui nous a directement fait accepter l'idée. Voulant faire plus que cela, nous nous sommes mis en tête d'allier exploration et énigme, ce qui rendrait le jeu plus vivant, tout en mettant en pratique la capacité de résoudre un casse-tête par le joueur. L'idée nous est venue après avoir dialogué entre nous au cours de l'année où nous nous sommes rendus compte que nous avions des objectifs similaires par rapport à ce projet de fin d'année. Ainsi, ayant tous été affectés par des jeux tels que Journey, Sky ou bien Professeur Layton, nous avons décidé de faire un jeu leur ressemblant et y prenant inspiration que ce soit pour des aspects de gameplay, d'ambiance ou de réalisation, tout en y apposant notre marque et des idées personnelles. Ce projet a donc des inspirations relativement nombreuses qui permettent à des néophytes en matière de création de jeux vidéos comme nous d'avoir un cadre assez précis en tête et qui permettra d'exprimer au mieux les idées que nous avons à coeur d'y ajouter.

1.1.2 Objet d'étude

Le but est de créer une expérience vidéoludique dont nous serons fiers et satisfaits. Le maniement d'un moteur de jeu de haut niveau comme Godot nous fera découvrir une nouvelle "manière" d'aborder la programmation par rapport à ce à quoi nous sommes habitués. En effet, aucun de nous n'a eu l'occasion d'utiliser ce genre de moteurs (Godot mais aussi Unity, Unreal engine, etc.) qui place la programmation uniquement dans les scripts, avec des outils très visuels de conception. De plus, l'utilisation de l'interface de création et d'animation 3D nous permettra d'augmenter individuellement nos compétences de design et de modélisation, qui resteront un atout dans le futur. Enfin, travailler en équipe dans un but commun est toujours un bon entraînement, aux vues de l'environnement de travail d'un ingénieur, et garder tout au long du projet une bonne cohésion et une bonne productivité est une tâche difficile, plus facile à s'acquitter si le projet nous passionne tous, en tant que joueurs de jeux vidéos avides de créer leur propre expérience. Godot permet de développer des jeux 2D, 3D mais également des solutions en réalité virtuelle ou réalité augmentée. Le logiciel étant ouvert (open source, à la différence de unity par exemple), nous pourrons développer nos propres modules que nous pouvons greffer au logiciel. Du point de vue du game design, nous recherchons un jeu qui ne se déroule pas avec une histoire très touchante ou narrativement parfaite, mais plutôt par un périple aux penchants poétiques avec un gameplay plus proche de la contemplation et de la réflexion que d'un TPS nerveux comme on en voit tant. Pour ce qui est du gameplay, le jeu sera en 3D et à la 3e personne, sans combats ni game over, mais plutôt basé sur de la réflexion et l'exploration du lieu, avec des énigmes à résoudre pour avancer etc. Depuis une dizaine d'années déjà, les jeux sans notions de niveaux à réussir, d'ennemis à combattre ou de game over se font de plus en plus nombreux, ils sont très souvent basés sur leur histoires, comme Life Is

Strange (2015), Until Dawn (2015) ou What remains of edith finch (2017). Pour les jeux sans réels combats ou game over mais avec plus de gameplay et notamment de l'exploration et de la réflexion, on retrouvera des titres comme Journey, Sky ou Professeur Layton. Ce genre de jeux qui résiste un peu au joueur sans chercher à le tuer nous intéresse tous et peut apporter des éléments de gameplay intéressants, comme de l'artisanat (mélanger des objets pour en créer un nouveau qui permet d'avancer dans le jeu), de possibles choix qui pourront faire changer le déroulement du jeu et notamment la fin, ou encore des secrets qui ne seraient pas obligatoires pour le finir mais qui rajoutent du contenu.

1.1.3 État de l'art

Le jeu faisant office de première vraie expérience 3D est BattleZone, un jeu développé en 1980 par Atari sur bornes d'arcades, il s'agit d'une simulation de batailles de tanks où les volumes sont représentés par des traits verts sur fond noir. Cependant, ce n'est que plus tard, dans les années fin 90 à 2000 que ce type de jeu vidéo apparaît tel qu'on le connaît. Notre jeu s'inspire en beaucoup de points (gameplay, ambiance, graphismes, etc.) de titres comme Journey (2012), Abzû (2016) ou Sky (2019), qui sont assez proche car tous basés sur la recherche d'un but lointain par une exploration lente et ayant tous le même style graphique épuré et environnements ouverts et immenses. Nous puisions également nos inspirations dans des jeux au gameplay assez différent comme Unravel (2016) ou Ori and the blind forest (2015). Ces deux jeux sont des jeux de plateforme, respectivement de réflexion et d'aventure. Tous ces titres ont une ambiance assez poétique et relaxante, ce qui ne les empêche pas d'avoir des moments d'action difficiles et nerveux comme dans Ori ou des moments de réflexion intenses comme dans Unravel. Dans ces jeux l'histoire est souvent simple et ne prend pas une place très importante, avec un but vague présenté dès le début et beaucoup d'exploration, plus ou moins guidée et linéaire pour y parvenir. La majorité de l'expérience réside donc dans le gameplay, le ressenti et l'ambiance générale. Le joueur peut donc s'y plonger et complètement s'évader de la réalité pendant son temps de jeu. Dans une moindre mesure, uniquement pour la partie mystère et énigmes, nous avons aussi des inspirations de la série des Professeur Layton (2007 - 2013), qui fait office de référence dans le domaine.

2 Découpage du projet

2.1 Avancement des tâches

Ce tableau montre la répartition actuelle des tâches.

Répartition	Jonathan	Alexandre	Ilias	Benoît
Menu		X		
Interface joueur	X			
Graphismes	X		X	X
Soundtrack	X	X	X	
Enigmes			X	
Personnage	X			
Sauvegarde		X		
Level design	X	X	X	X
Story telling	X	X	X	X
Site	X		X	

Ce tableau montre l'avancée actuelle des différentes tâches du projet.

Tâches	Avancement réel	Avancement prévu
Menu	90	100
Interface joueur	100	100
Graphismes	100	100
Soundtrack	100	100
Enigmes	90	100
Personnage	100	100
Sauvegarde	100	100
Level design	100	100
Story telling	100	100
Site Web	100	100

3 Avancement

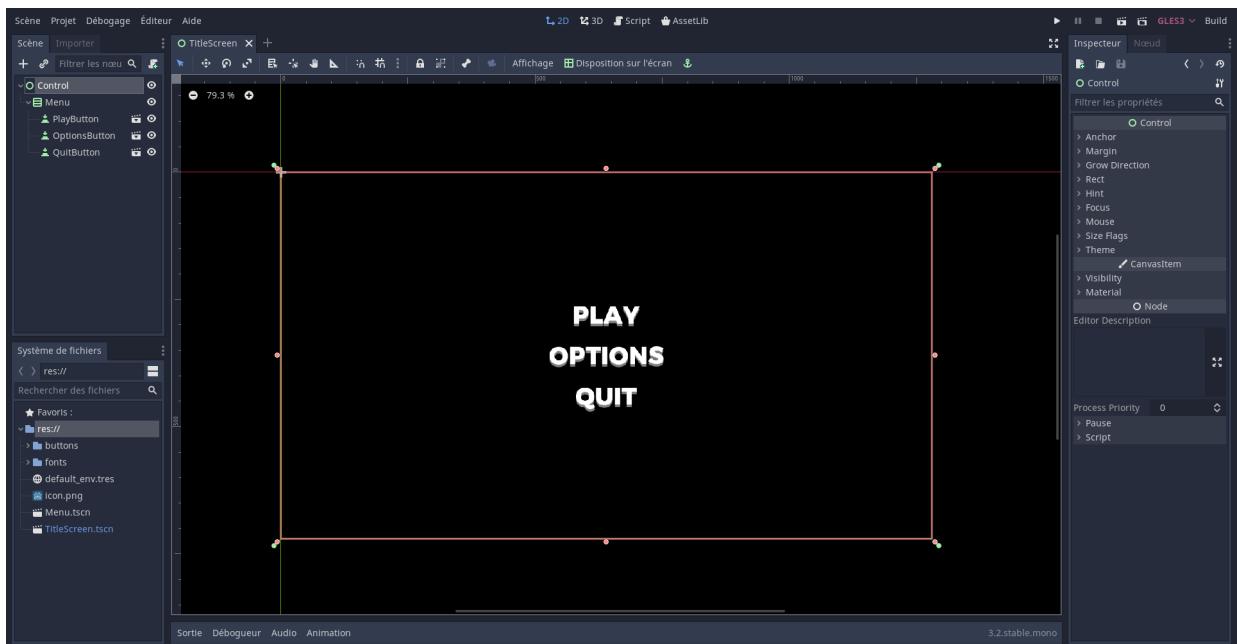
Chaque fonctionnalité du projet est détaillée ci-dessous dans leurs sections respectives. Ces sections correspondent aux catégories de fonctionnalités que nous avions mentionnées dans le cahier des charges. Pour chaque section, le développement de chaque fonctionnalité, ainsi que tous les aspects qui la composent, seront détaillés. Pour chaque fonctionnalité qui s'y prête, nous préciserons leur fonctionnement et leurs enjeux techniques.

3.1 Interface

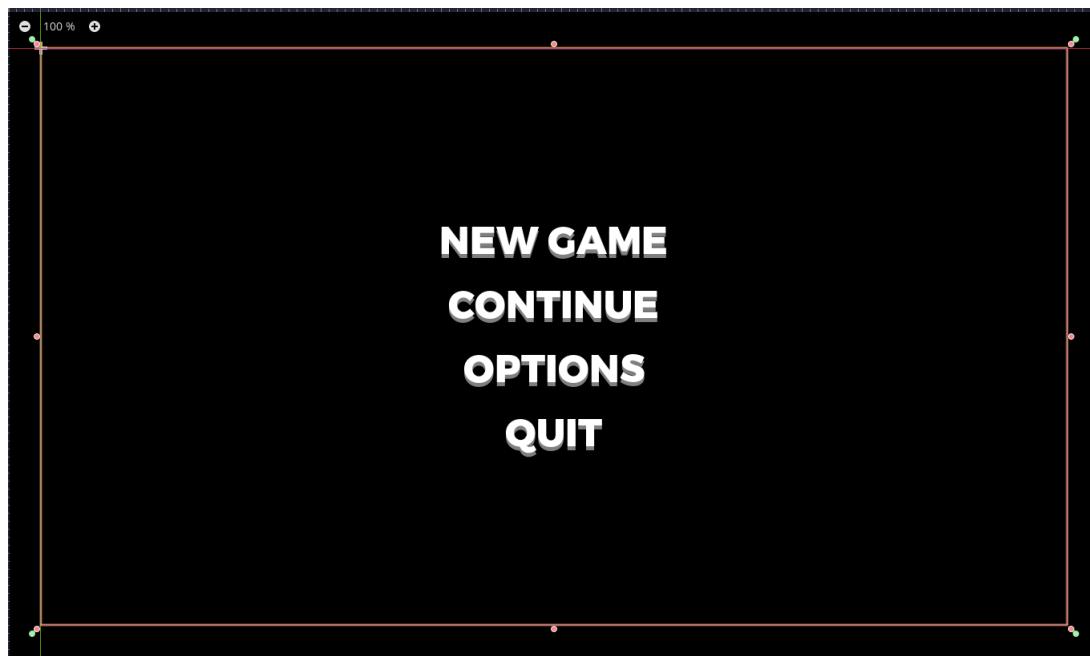
L'interface est ce qui relie le jeu au joueur, c'est via celle-ci que le joueur peut naviguer à travers les menus afin de lancer la partie, continuer le jeu, quitter le jeu ou tout simplement pour sauvegarder sa partie en cours. Nous retrouvons donc l'interface à tous les niveaux du jeu pour qu'à n'importe quel instant, le joueur puisse sauvegarder et ne pas se retrouver bloquer quelque part. Cette accessibilité permet de créer une expérience de jeu plus agréable et personnalisée pour chaque joueur.

3.1.1 Menu principal (Alexandre M.)

Au départ, le menu était quasiment vide. Il a bien évolué depuis le début de notre projet. Seulement, nous n'avons pas implémenté d'options par manque de temps. En revanche, nous avons réussi à mettre en place un système de sauvegarde présenté plus loin dans ce rapport. Nous avons commencé avec seulement trois boutons sans aucun script implémenté.



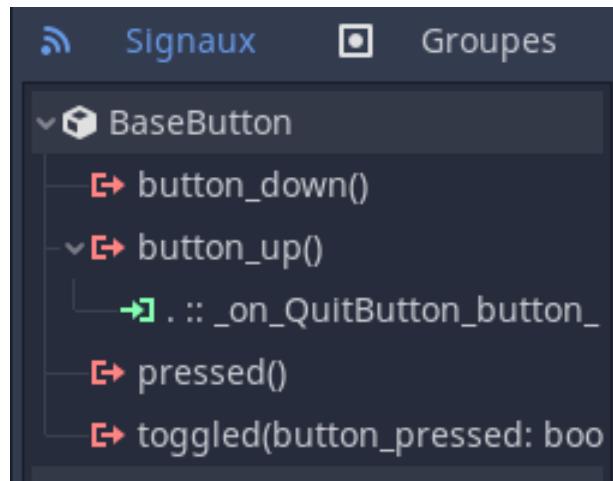
Ensuite nous avons choisi de séparer le bouton "PLAY" en deux boutons différents nommés "NEW GAME" et "CONTINUE". Cela permet d'accéder directement au jeu sans avoir à entrer dans une autre fenêtre avant de jouer car autrement, le bouton "PLAY" nous enverrait dans cette fenêtre permettant de choisir entre les deux autres boutons et cela serait une redirection inutile avant le jeu.



Ensuite, nous avons intégré au menu une image servant d'arrière plan pour ne pas le laisser vide.



Dans le menu, plusieurs signaux ont été utilisés. En voici un exemple :



Ces signaux sont intégrés dans la partie la plus importante du menu, le code en C#. Premièrement, le bouton "QUIT". Pour cela il a suffit d'appeler la fonction Godot qui permet de quitter le jeu.

```
1  using Godot;
2  using System;
3
4  public class QuitButton : Node
5  {
6      private void _on_QuitButton_button_up()
7      {
8          GetTree().Quit();
9      }
10 }
```

Deuxièmement, le bouton "NEW GAME". Celui-ci, créant une nouvelle partie, sert à écraser la dernière sauvegarde en réécrivant les valeurs de départ avant d'ouvrir la scène du jeu.

```

1  using Godot;
2  using System;
3  using System.IO;
4
5  public class NewGameButton : Node
6  {
7      private void _on_NewGameButton_button_up()
8      {
9          StreamWriter sw = new StreamWriter("save");
10         sw.WriteLine("-83,64561\n12,40903\n86,57724\n0000000000");
11         sw.Close();
12         GetTree().ChangeScene("res://PlayerTest.tscn");
13     }
14 }
```

Troisièmement, le bouton "CONTINUE". Ce bouton continue tout simplement la partie. Il n'a pas à toucher à la sauvegarde car elle est load dans la fonction "Ready" du fichier "Game.cs". Il sert donc seulement à afficher le jeu.

```

1  using Godot;
2  using System;
3  using System.IO;
4
5  public class ContinueButton : Node
6  {
7      private void _on_ContinueButton_button_up()
8      {
9          GetTree().ChangeScene("res://PlayerTest.tscn");
10     }
11 }
```

Enfin, il y a le code principal du menu qui sert à afficher ou masquer l'écran de chargement (affiché dans la partie 3.1.3) et la souris. Il permet également de recadrer le menu dans la fenêtre du jeu.

```

public override void _Ready()
{
    GetNode<TextureRect>("Loading").Visible = false;
    Input.SetMouseMode(Input.MouseMode.Visible);
    Vector2 viewport = new Vector2(GetViewport().Size.x, GetViewport().Size.y);
    this.RectScale = new Vector2(2,2);
    this.SetPosition(viewport/2);
}

private void _on_NewGameButton_button_down()
{
    GetNode<TextureRect>("Loading").Visible = true;
}

private void _on_ContinueButton_button_down()
{
    GetNode<TextureRect>("Loading").Visible = true;
}

```

3.1.2 Menu in-game (Ilias B. et Jonathan P.)

Afin que le joueur puisse sauvegarder, retourner au menu ou aller dans les options, un autre menu que le principal est nécessaire, un menu “in-game”. Celui-ci s’affiche lorsque le joueur appuie sur échap rendant le menu visible et nous appelons la fonction qui gère le menu. Il est placé dans la scène du personnage, de sorte à toujours faire face à la caméra. On navigue à l’intérieur avec les contrôles pour avancer et reculer le personnage (w et s), et entrée sert à valider le choix. Afin de pouvoir afficher ce menu, il a fallu simuler de la 2D dans un monde en 3D. Pour que cela soit plus simple, tous les procédés sont encapsulés dans une scène de type simple : spatial. Le type dont dérivent tous les objets 3D. Cette scène est donc composée de deux grandes parties : la partie qui “existe” dans le monde en 3D, et la partie qui affiche du 2D. L’objet Viewport permet d’afficher un écran dans un monde en 3D, ici cet écran montre le menu. Les objets fils du Viewport sont les éléments à afficher. Le “Quad” est une mesh (un objet avec une forme 3D) et cela permet d’avoir un support pour le Viewport. Ces deux objets ensemble permettent donc d’afficher de la 2D dans un monde en 3D. La dernière chose à prévoir est ce qui va permettre à l’utilisateur de naviguer dans le menu. Pour cela, deux flèches se placent de chaque côté du bouton actuellement sélectionné et la touche entrée déclenche une action différente :

- Resume : ferme le menu
- Save : sauvegarde la partie
- Options : ouvre le menu des options (non implémenté)
- Quit : retourne au menu principal

Nous avons dû changer le menu in-game depuis sa première version, devenant illisible lorsque placé dans le jeu à cause des couleurs qui n’étaient pas très visibles. De plus, nous avions effectué une erreur d’inattention en mettant de l’anglais et du français dans le menu.



3.1.3 Ecran de chargement (Ilias B. et Alexandre M.)

Le temps d'attente entre le moment où le joueur est dans le menu puis se retrouve dans le jeu nous semblait assez long. Ainsi, nous avons voulu créer un écran de chargement assez beau esthétiquement parlant afin de pallier la frustration que pourrait causer ce délai. Nous avons donc mis en place une lune entourée d'esprits, rappelant le côté mystérieux de ce jeu.



3.2 Map

3.2.1 Modélisation (Benoit B.)

Pour la modélisation de la map, nous avons donc, au début, décidé de la faire à partir de Blender et de l'addon “Blender ESCN exporter” qui nous permettrait d'exporter tous nos objets 3D sur Godot, et dans notre cas, la map.

Cependant, nous avons rencontré beaucoup de difficultés quant à l'élaboration de celle-ci sur Blender, donc nous avons cherché d'autres solutions.

Après de nombreuses recherches, nous avons remarqué qu'il existait un logiciel de création de map 3D très bien optimisé et qui nous permettait de l'exporter sur tous les moteurs de jeu ainsi que sur Blender. Ce logiciel se nomme World Machine.

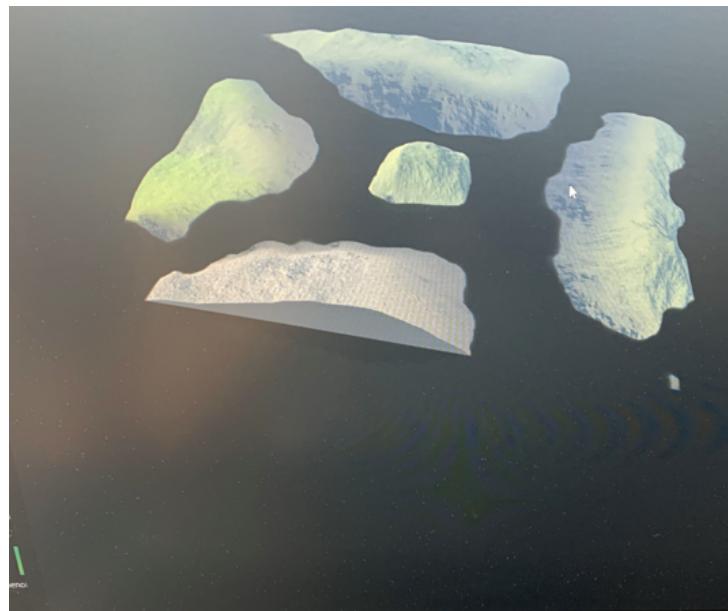
Contrairement aux éditeurs de terrain traditionnels, World Machine utilise une approche procédurale. En effet, celui-ci peut former le terrain en utilisant des blocs de construction de base tels que des fractales.

Nous pouvions aussi appliquer des textures à nos différentes îles assez aisément après avoir formé celles-ci.

Pour la modélisation de la map, qui dans le domaine des jeux vidéo représente le terrain sur lequel le personnage va réaliser toutes ces actions ainsi que son aventure.

Pour commencer à la modéliser, lors de la période avant la seconde soutenance, nous avons utilisé le logiciel World Machine qui est un logiciel largement utilisé par des modélisateurs de terrain pour des jeux vidéo en 3D ou autres.

Après avoir passé un certain temps pour comprendre comment le logiciel fonctionnait, nous avons utilisé le logiciel afin d'obtenir la map que vous pouvez voir ci-dessous :

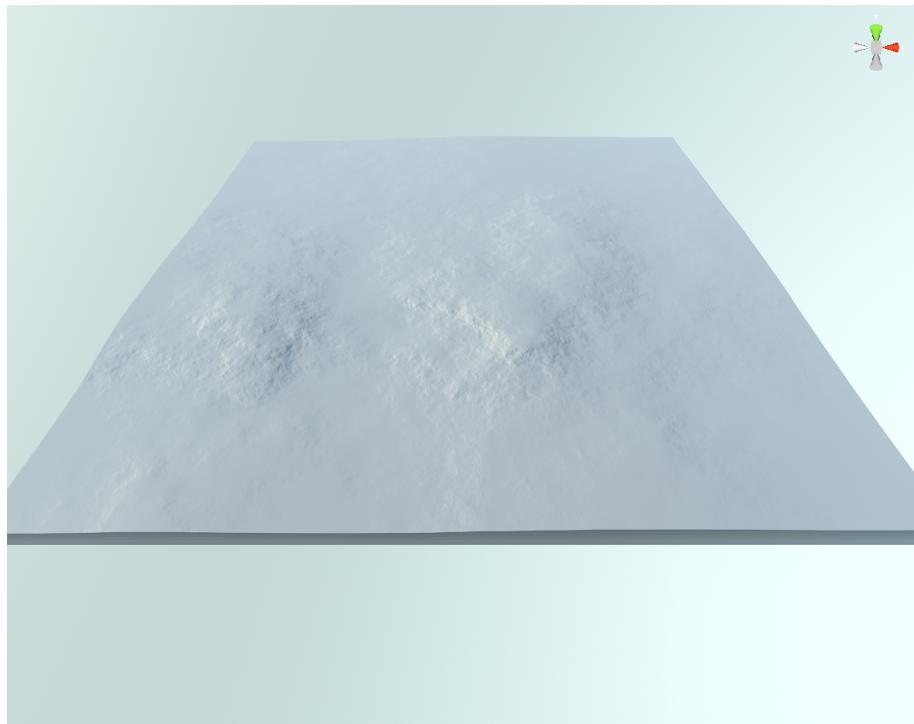


Pour l'élaboration de celle-ci, nous avons surtout utilisé l'outil “Terrain Shape Designer” pour éléver ou abaisser rapidement le terrain à des endroits spécifiques en temps réel, ce qui nous a permis de créer les cinq îles que vous pouvez voir ci-dessus. Nous avons aussi utilisé l'ajout de textures à un paysage et de l'ajustement de certains paramètres pour répartir les textures sur le paysage de manière réaliste, ainsi que l'utilisation des zones, appelées “Area” dans le logiciel que nous avons utilisé de multiples façons avec notre terrain et qui nous a permis de modifier le terrain seulement aux endroits que l'on souhaitait.

Sachant que la longueur et la largeur du terrain étaient par défaut de 2524 par 2524 mètres.

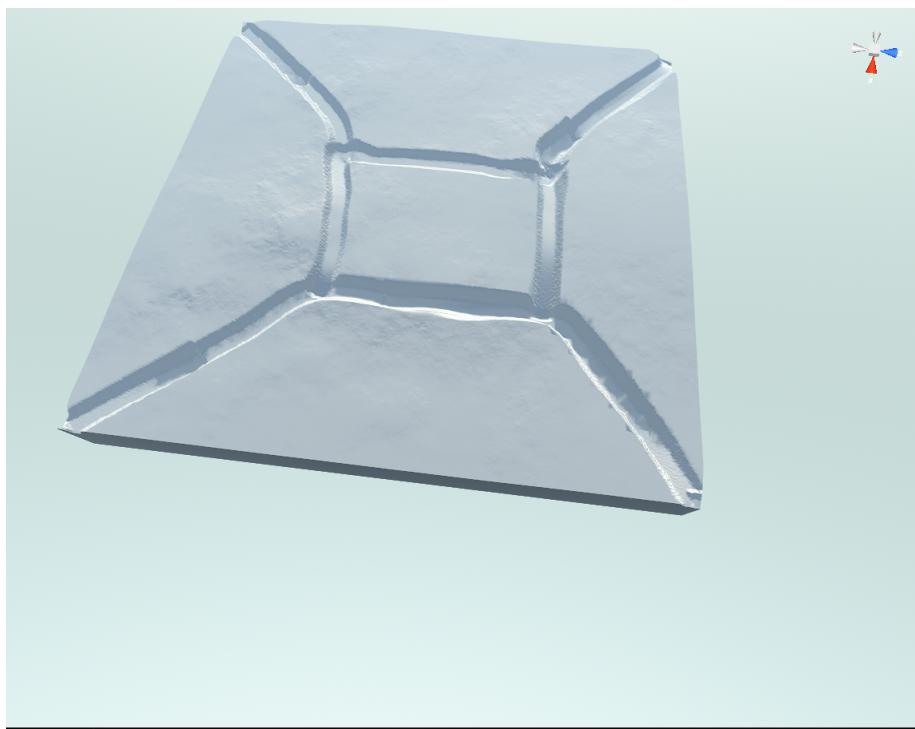
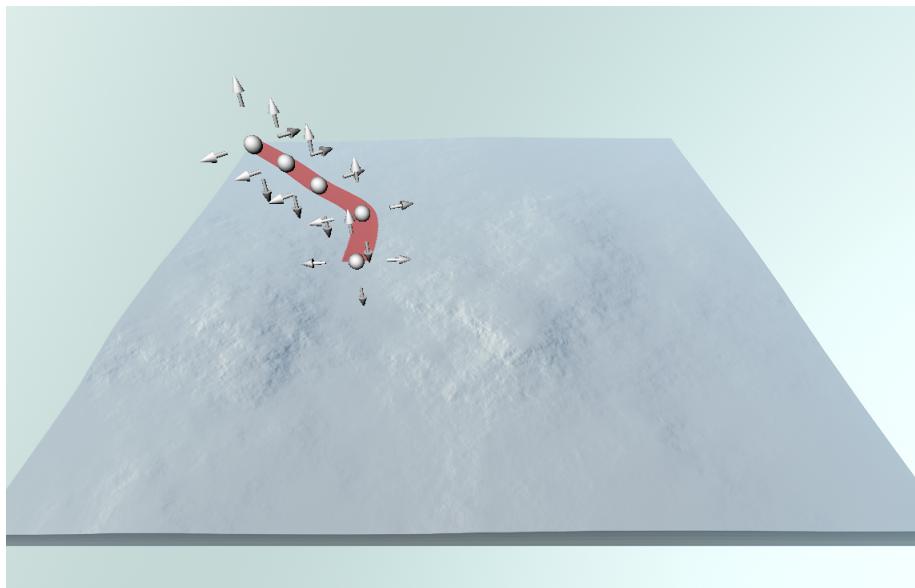
Cependant, après s'être concerté entre nous, nous avons remarqué que la distance qu'il y avait entre les îles étaient trop grandes et il fallait donc modifier la map au plus vite afin qu'elle soit exploitable.

Pour cela, il existait un paramètre sur le logiciel, nommé flatten, qui permet d'aplatir le terrain au niveau de la surface, afin d'obtenir ce que nous pouvons voir ci-dessous :



Ensuite, nous avons appliqué à ce terrain un layer qui s'appelle “path” et qui nous permet de créer des rivières.

Avec ce layer, nous avons pu mettre des points sur le terrain qui vont se relier et qui forment un chemin où nous pouvons modifier l'épaisseur et la largeur, ainsi que la position de chaque points afin de former des contours précis où pourrait passer la mer, et donc donner l'impression au joueur de se croire dans une île.



Ainsi, nous avons obtenu le terrain que vous pouvez voir ci-dessus qui est exploitable car la distance entre les îles était telle qu'on pouvait mettre des ponts pour que le joueur puisse voyager d'une île à une autre tout en profitant de la grandeur de la map.

Une fois le terrain modélisé, nous devions trouver une texture qui conviendrait le mieux à l'idée qu'on avait de la map.

La map étant exactement comme on le souhaitait, nous devions ensuite exporter cette map sur Blender qui est un logiciel entièrement tourné vers la création 3D et dont l'une de ses originalités est d'appliquer des textures de son choix aux modèles créés.

Pour exporter la map, le logiciel nous a permis de mettre notre map sous le format .obj, qui était un format exploitable par Blender.

Ensuite, nous lançons Blender et nous allons dans [File] et [import], puis nous choisissons le format du fichier que l'on souhaite importer.

Une fois dans Blender, les textures permettent l'ajout de détails aux surfaces modélisées. Le placage de textures consiste en la projection d'images ou de motifs sur ces surfaces. Les textures peuvent être également le fruit de formules mathématiques. L'application de texture à la surface des objets créés affecte plusieurs paramètres dont la couleur, la transparence et la réflexion. Pour appliquer une texture sous Blender, nous devions créer un matériau (nommé "Material" sur Blender) que nous appliquons à un objet 3D, qui est ici la map, puis nous choisissons l'image qui sera utilisée comme texture et nous la copions dans le répertoire courant de nos travaux Blender de manière à appliquer la texture au matériau et donc à l'objet.

Pour la texture, nous avons d'abord décidé de prendre une texture verte trouvée sur le web, puis nous l'appliquons sur l'ensemble de la map de manière à ce que cela donne une impression d'herbe au joueur.

Ensuite, il fallait exporter la map avec les textures de Blender vers Godot. Pour cela, après avoir demandé des renseignements sur des serveurs Discord, sachant qu'il était difficile d'exporter un gros objet 3D de Blender vers Godot, on nous a conseillé d'utiliser un addon appelé "escn_exporter" qui est disponible sur la documentation de Godot Engine.

Création de la mer :

Une fois le terrain créé avec les textures et une fois exporté sur Godot, il a fallu s'occuper de l'élaboration de la mer.

Sachant que la map a été faite de manière à ce que nous puissions facilement créer des cours d'eau passant entre les différentes îles.

Donc, pour la création de la surface d'eau, nous avons d'abord commencé à faire un noeud avec la racine et une « MeshInstance » .

MeshInstance est un nœud qui prend une ressource Mesh et l'ajoute au scénario en cours en créant une instance de celle-ci. C'est la classe la plus souvent utilisée pour obtenir un rendu géométrique en 3D et peut être utilisée pour créer une instance d'un seul Mesh à plusieurs endroits.

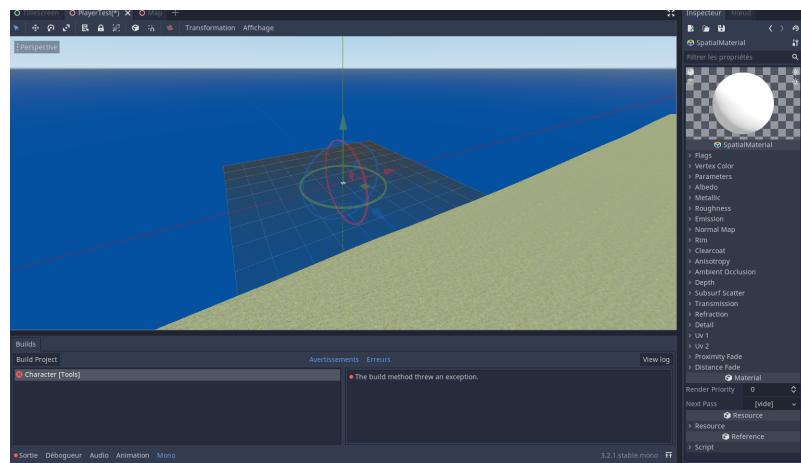
Ensuite, nous devons configurer notre Mesh en tant que « PlaneMesh », qui est une classe représentant un PrimitiveMesh planaire. Cette maille plane n'a pas d'épaisseur. Par défaut, ce maillage est aligné sur les axes X et Z.

Pour cela, nous avons dû régler la taille de ce Mesh à 20000 pour les coordonnées x et y de manière à ce que cet objet recouvre la map.

Ensuite, pour s'occuper de la texture et de l'apparence de ce Mesh, nous devions aller sur « PrimitiveMesh » et créer un nouveau « SpatialMaterial » .

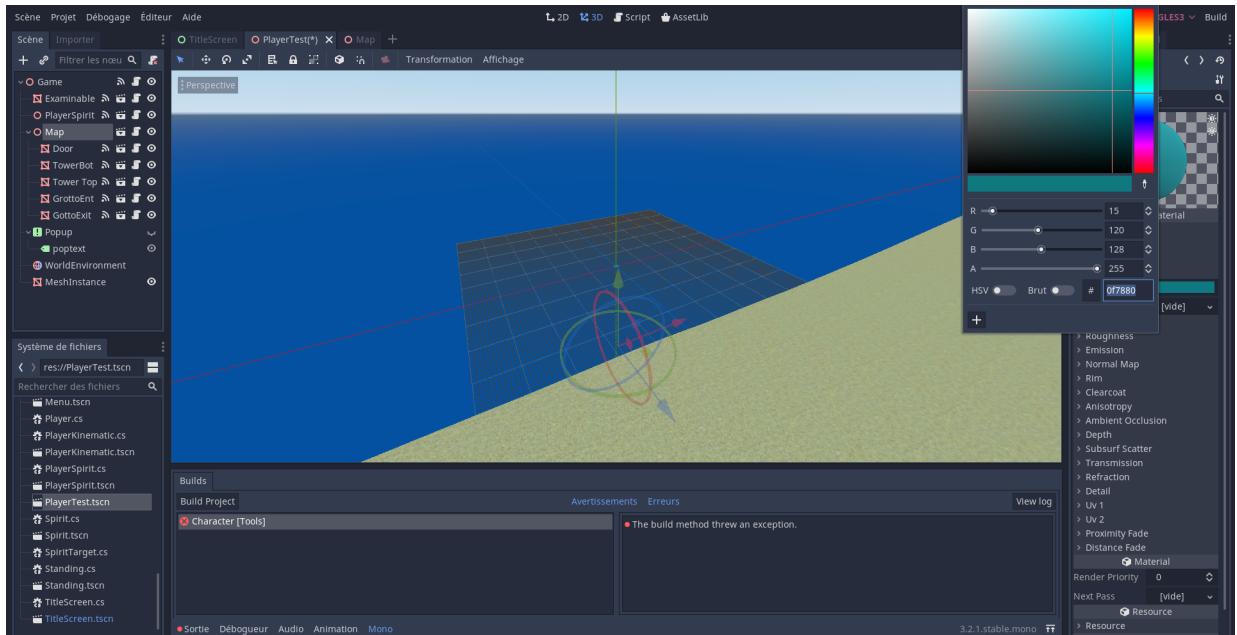
Nous avons créé un SpatialMaterial car celui-ci est un matériau 3D par défaut qui vise à fournir la plupart des caractéristiques que les artistes recherchent dans un matériau, sans qu'il soit nécessaire d'écrire du code de shader.

Une fois ceci fait, nous avons accès à tous les paramètres que vous pouvez voir ci-dessous :



Après avoir eu accès aux paramètres, nous avons pu modifier la couleur de la Mesh, donc de l'eau grâce au paramètre nommé “Albedo”.

Avec ceci, avec taton, nous avons décidé d'avoir comme couleur celle que vous pouvez voir ci-dessous :



Une fois ceci fait, nous avons remarqué que nous pouvions donner à l'eau un effet plus brillant par rapport à la lumière du jour, pour cela, nous devons aller sur “Roughness”, qui affecte principalement la façon dont la réflexion de la lumière se fait. Une valeur de 0 en fait un miroir parfait tandis qu'une valeur de 1 brouille complètement la réflexion (simulant le micro-surfâge naturel). La plupart des types de matériaux courants peuvent être obtenus grâce à la bonne combinaison de “Metallic” et de Roughness.

Ainsi, dans notre cas, nous avons dû augmenter ce paramètre.

Ensuite, nous avons utilisé “Normal Map” qui permet de définir une texture qui représente un détail de forme plus fin. Cela ne modifie pas la géométrie, mais seulement l'angle d'incidence de la lumière. Dans Godot, seuls R et G sont utilisés pour les cartes normales, afin d'obtenir une meilleure compatibilité. Grâce à cela, nous avons pu créer des petites vagues à la surface de l'eau en activant Normal Map, en lui appliquant la même texture que l'eau et en lui mettant comme dimension (“scale” sur Godot) 0.02.

Pour finaliser la création des petites vagues, nous avons aussi dû utiliser “UV 1”, car Godot supporte 2 canaux UV par matériau. Les UV peuvent être échelonnés et décalés, ce qui est utile pour les textures à répétition, et dans notre cas, les vagues.

Ensuite, nous activons ce qui est appelé “Refraction”. Lorsque la réfraction est activée, elle remplace le mélange alpha, et Godot tente plutôt d’aller chercher l’information derrière l’objet rendu. Cela permet de déformer la transparence d’une manière similaire à la réfraction.

Dans notre cas, nous avons gardé le scale bas car nous travaillons dans une grande surface.

Nous avons d’ailleurs utilisé “Proximity Fade”, car grâce à celui-ci, Godot permet aux matériaux de s’effacer en fonction de la proximité les uns des autres ainsi qu’en fonction de la distance qui les sépare du spectateur. La “Proximity Fade” (décoloration par proximité) est utile pour des effets tels que des particules molles ou une masse d’eau se fondant doucement dans les rives.

Sachant que nous avons paramétré la distance dont s’applique “Proximity Fade” à 1.

Nous avons aussi utilisé “Distance Fade” qui est utile pour les puits de lumière ou les indicateurs qui ne sont présents qu’après une distance donnée. Cela est donc très utile pour assurer une transition en douceur entre la vue sur l’eau et la vue sous l’eau.

Enfin, pour finaliser la création de notre eau, nous avons fait le choix d’ajouter une animation à notre eau.

Pour ce faire, nous avons créé un noeud à la Mesh que nous avons créé avec un “AnimationPlayer”.

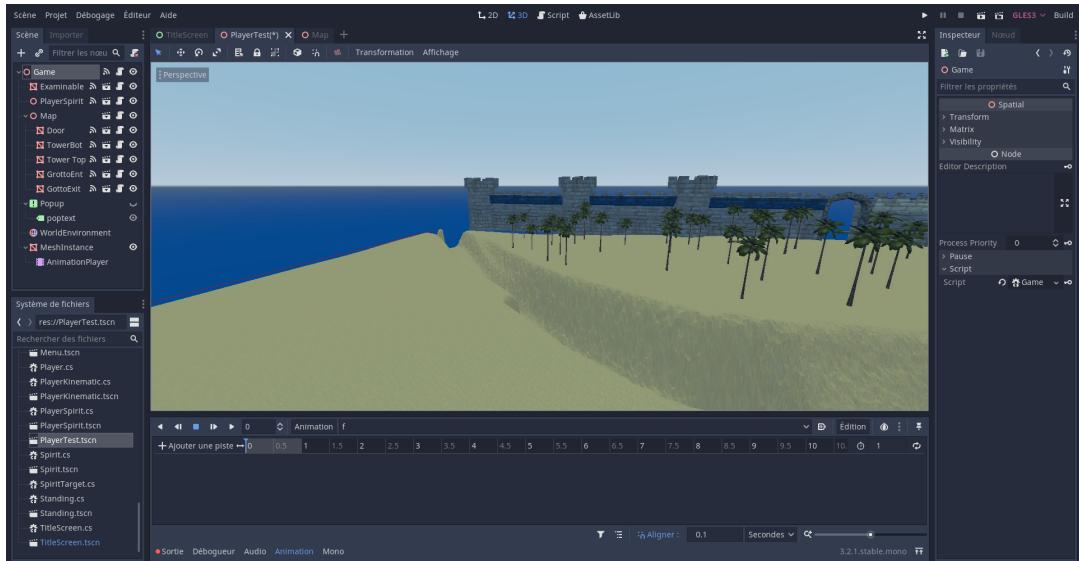
Puis nous créons une nouvelle animation, et nous animons l’uv pour déplacer la texture normale/réfractée qui peut être retrouvée dans “UV 1” que nous avons utilisé précédemment.

Pour animer l’UV, nous avons configuré le “Offset” à (1, 0, 0).

Enfin, nous revenons sur l’animation que nous avons créé et nous n’oublions pas d’activer la “Lecture automatique au chargement” et le “Bouclage de l’animation”. Cela servira notamment à ce que l’animation qui est donnée à notre eau soit en continue.

Pour que cette animation paraisse encore plus réelle, nous avons pu ajuster la vitesse de l’animation de l’eau à 0.5.

Après avoir réalisé tout ce que nous avons dit ci-dessus, nous avons obtenu la mesh que vous pouvez voir ci-dessous représentant notre mer :



Elaboration de la grotte :

Après avoir réalisé le terrain de notre map, on s'est occupé des différents modèles de la map, comme la tour, ou bien le labyrinthe que nous avons réalisé en mettant bout à bout le même modèle d'un petit labyrinthe et qui nous a permis d'en obtenir un plus complexe.

Cependant, pour ce qui est de la grotte, nous n'avons malheureusement trouvé aucun modèle 3D sur des sites internet comme nous avions pu le faire pour le labyrinthe ou la tour, nous avons donc dû trouver une autre alternative.

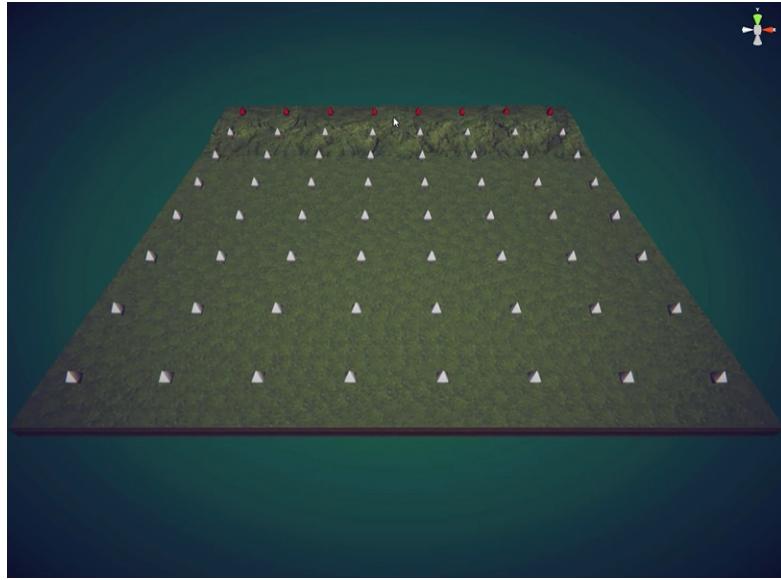
Celle-ci consistait à prendre le même logiciel de création de terrain utilisé pour la modélisation de la map, c'est-à-dire WorldMachine.

Néanmoins, nous nous sommes vite rappelé que le logiciel ne pouvait pas “creuser” de façon verticale ou en latérale. Nous ne pouvons donc pas former nos galeries directement depuis le logiciel.

Nous avons donc eu l'idée d'utiliser le “Path Filter” proposé par World Machine. Le filtre de “Path” est un filtre de conception qui peut être utilisé pour créer des montagnes, des rivières, des pistes, des chemins, des routes simples, et tout ce que vous pouvez faire avec un chemin vectoriel typique.

Ainsi, nous pouvions créer des rivières et quand nous retournons notre map sur Godot dans l'autre sens, nous obtenons notre grotte.

Nous avons donc commencé à prendre un terrain plat qui était assez épais au niveau du sol afin de pouvoir creuser assez profondément. Pour cela, nous avons créé un nouveau terrain où nous avons appliqué le filtre “Flatten” qui a rendu le terrain plat, puis nous avons augmenté l'épaisseur de notre terrain avec l'outil “Terrain Shape Designer” pour éléver ou abaisser rapidement le terrain à des endroits spécifiques en temps réel. Vous pouvez voir ci-après l'utilisation que nous pouvons avoir de ce filtre.



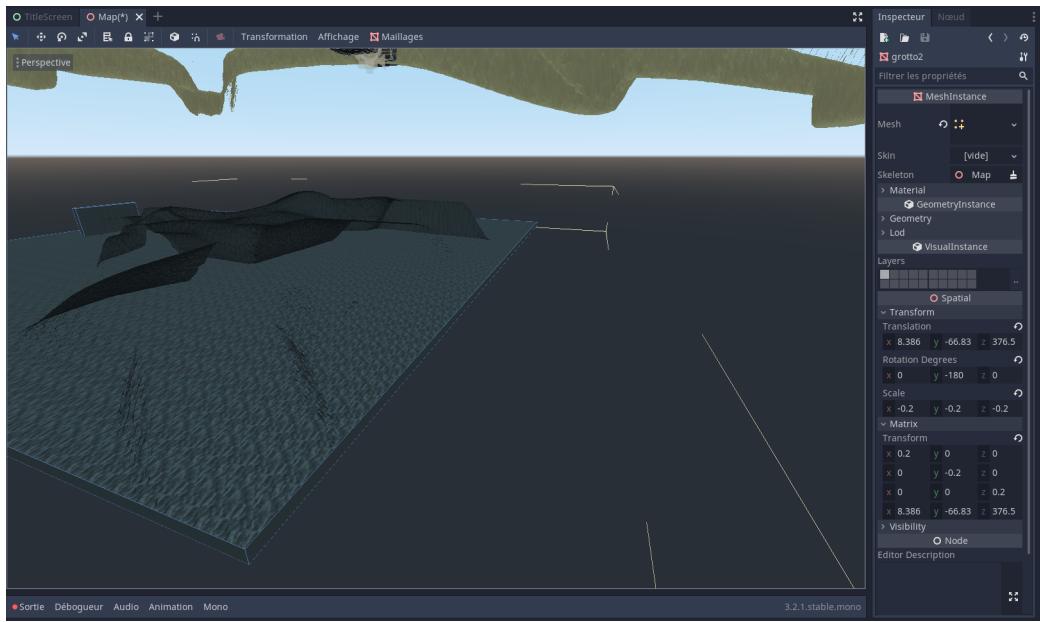
Après avoir élevé notre terrain, nous nous sommes mis d'accord avec la forme de notre grotte, et nous avons donc réalisé un brouillon afin de savoir exactement où nous devons utiliser le filtre “Path”.

Ensuite, lorsque nous avons commencé à réaliser notre grotte, nous devions faire attention à la largeur et à l'épaisseur de nos galeries qui devaient avoir les mêmes dimensions hormis la pièce centrale de la grotte qui devait être un peu plus grande et profonde que les galeries.

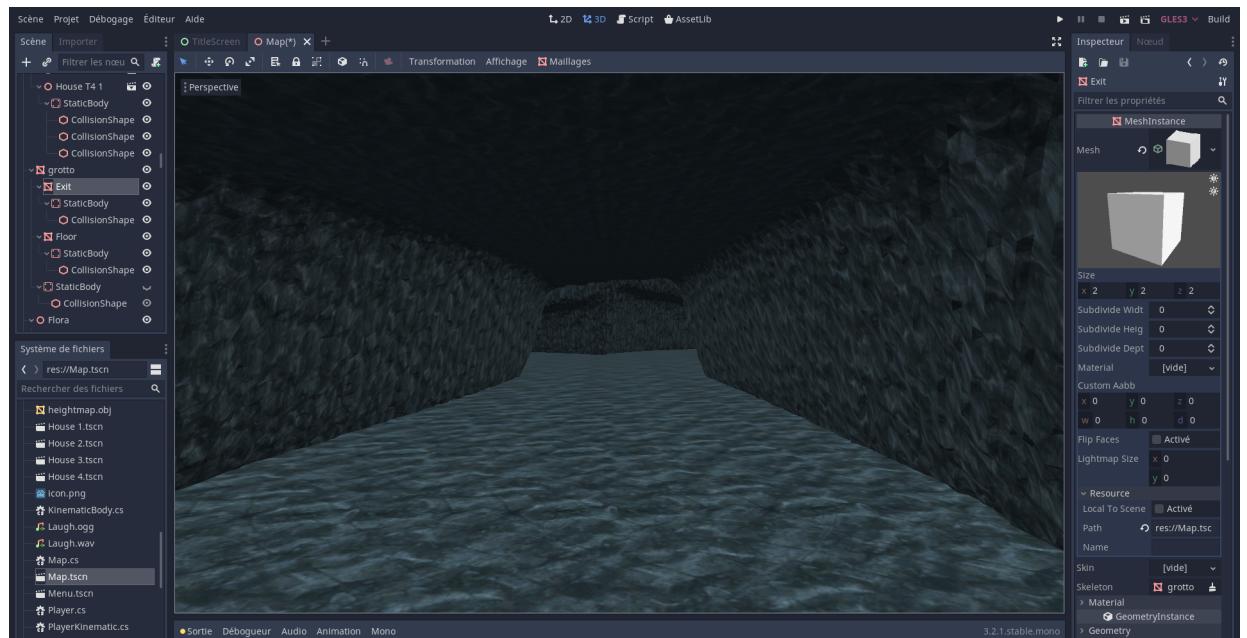
Une fois que nous avons réalisé nos galeries, nous exportons le terrain vers Godot de la même manière que pour la map, c'est-à-dire que nous devons l'exporter depuis le logiciel sous le format .obj, nous mettons ce fichier dans le même dossier que celui du projet, puis celle-ci va directement s'importer en tant qu'objet ou en tant que scène.

Nous avons décidé de le prendre en tant que scène, puis nous créons un noeud avec la map (appelée “map.tscn”) et une MeshInstance qui prendra la forme de la map. Ensuite, sachant que la grotte était beaucoup trop grande pour pouvoir la placer sur la map, nous avons décidé de placer la grotte “en-dessous” de la map et d'utiliser un téléporteur afin que le personnage puisse y avoir accès, sachant que nous avons dû modifier les dimensions de la grotte qui était beaucoup trop grande pour le joueur. Pour cela, nous devons modifier le scale de la Mesh représentant la grotte.

Vous pouvez voir ci-dessous où est située la grotte par rapport à la map :



Enfin, pour finaliser la conception de notre grotte, il fallait que nous appliquions une texture à la grotte qui ne soit pas trop sombre pour le joueur et ni trop claire, ainsi, nous avons décidé d'utiliser une texture rocallieuse que nous avons trouvé sur le Web.



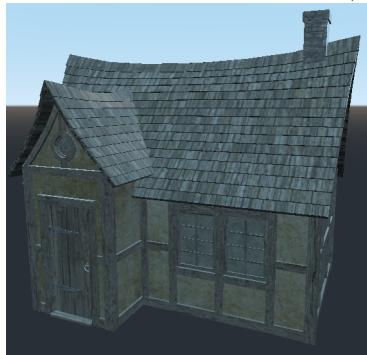
3.2.2 Objets et textures (Jonathan P. et Benoit B.)

Une grande partie du travail sur la carte a été de l'aménager, de la remplir avec les éléments nécessaires au jeu. Ces derniers ont été trouvés sur internet (voir sites en annexes) et retexturés grâce à Blender et Godot. En effet la grande majorité des modèles 3D s'importent sans texture appliquée. Il a donc fallu les réassigner à chaque partie du modèle et changer certaines propriétés. Le plus souvent, il faut réajuster "Roughness" qui gère le taux de réflexion de la lumière, "Alpha scissors" qui sert à appliquer une texture image. Cette dernière nous permet de retirer le fond, ce qui est surtout utile pour la végétation. Enfin, nous avons réajusté "UV1", qui est utilisé lorsque nous voulons répéter une texture comme le sable sur le sol ou le mur de la grotte par exemple.

Les modèles sont les suivants :

- La tour : elle trône au centre de la carte et est visible où que nous nous trouvions (sauf dans la grotte et les maisons évidemment). Elle est un peu différente des autres modèles 3D car elle n'a pas de texture "externe" mais est peinte directement sur le modèle. Elle est accessible après avoir réussi les 8 autres énigmes cachées sur la carte. À son sommet se trouve l'avant-dernière énigme qui permettra au joueur de débloquer l'accès à la dernière île avec le mur.
- Le labyrinthe : il donne accès au village, et est composé d'un labyrinthe très simple répété avec des rotations.
- Le village : il contient quelques maisons, de quatre types différents, dont une où l'intérieur est accessible. C'est cette partie du texturing qui a pris le plus de temps car les modèles sont relativement complexes et il y a un grand nombre de textures différentes. Il y a 3 énigmes cachés dans ce village.
- Le mur : il est composé d'une entrée, de tours et de murailles, et constitue le but final du jeu.
- L'entrée de la grotte : la carte et la grotte n'étant pas reliées, il fallait une structure comme entrée. Au centre de la carte se trouve un téléporteur permettant d'aller dans la grotte.
- La végétation : un modèle d'arbre et un modèle de buisson permettent de créer un semblant de forêt une fois dupliqués un grand nombre de fois.
- Les ponts : ils servent à pouvoir se déplacer entre les îles.

Exemples de modèles 3D (le reste se trouve en annexes) :



3.2.3 Examinables (Jonathan P.)

Les objets examinables sont les objets qui lanceront les énigmes. Ils peuvent prendre n'importe quelle forme étant donné qu'ils sont de type mesh 3D, et possèdent plusieurs capteurs : — une “area” qui capte quand le joueur est proche de l'objet — un “visibility notifier” qui capte si l'objet est proche du centre du champ de vision du joueur Si ces deux conditions sont vérifiées, un point d'exclamation apparaît au dessus afin de signaler au joueur qu'une action peut être réalisée.

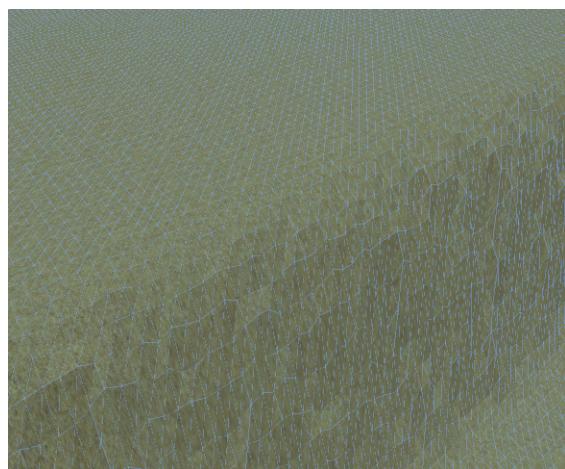
Ils servent dans le jeu à téléporter le joueur (du sommet au bas de la tour, à l'intérieur et à l'extérieur de la grotte ou de la maison), ou bien à ouvrir une des énigmes. Cependant, lorsque beaucoup d'objets d'une même instance ont des signaux de connectés, un problème de Godot fait que certains signaux sont interceptés par les mauvais listener et un comportement inattendu peut arriver. Il s'agit du problème principal non résolu du jeu.

3.2.4 Collisions (Jonathan P.)

Dans un jeu vidéo, la partie graphique, affichée à l'écran, et ce qui est considéré comme "objet physique" sont souvent très différents. La hitbox d'un personnage, par exemple, n'est pas pile son modèle 3D (ou son sprite en 2D), mais une forme simple lui ressemblant au mieux. La carte du jeu étant une forme complexe, la gestion des collisions avec cette dernière ne pouvait pas se faire avec une forme simple, plane par exemple. Sur Godot, il existe une option pour créer un corps de collisions très proche du modèle, sous la forme d'un solide avec une multitude de faces. Bien que cela soit très pratique et précis, cette approche est aussi très gourmande en place, c'est notamment cela qui est la cause des temps de chargements trop longs. Il y a des modèles qui sont beaucoup plus simples comme le personnage qui a une hitbox plus ou moins cylindrique ou encore la muraille composée de cylindres et de parallélépipèdes rectangles. Dans Pricefield Garden, toutes ces hitbox servent à délimiter des objets en vu de leur appliquer des collisions avec le personnage, ou bien à délimiter des zones pour que des zones qui envoient un signal déclenchent une fonction lorsque le personnage y entre.

Voici des exemples de HitBox (en fils bleu)

Corps trimesh :



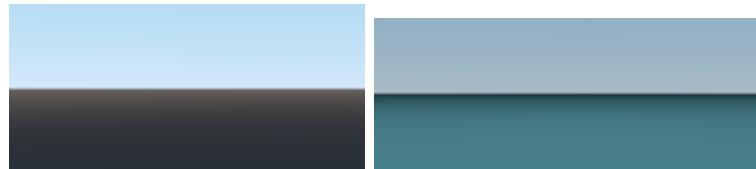
Collision simple :



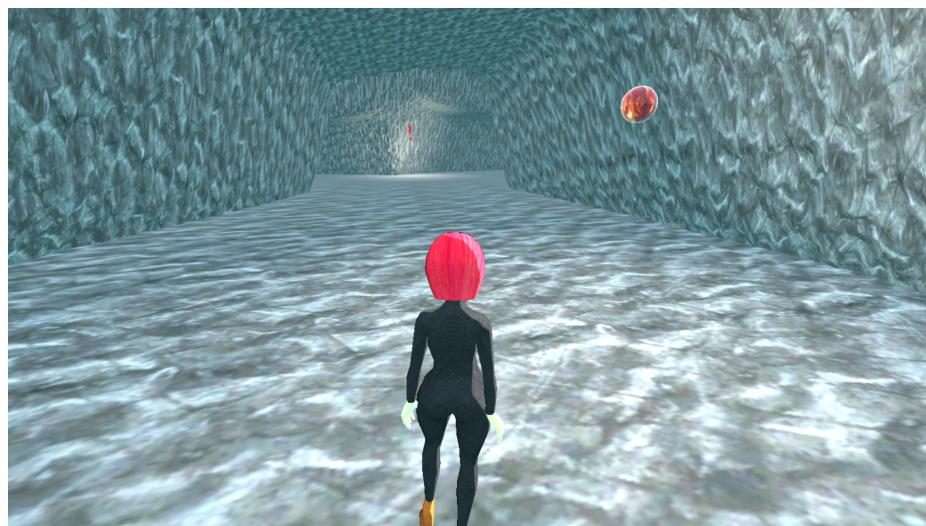
3.2.5 Environnement (Jonathan P.)

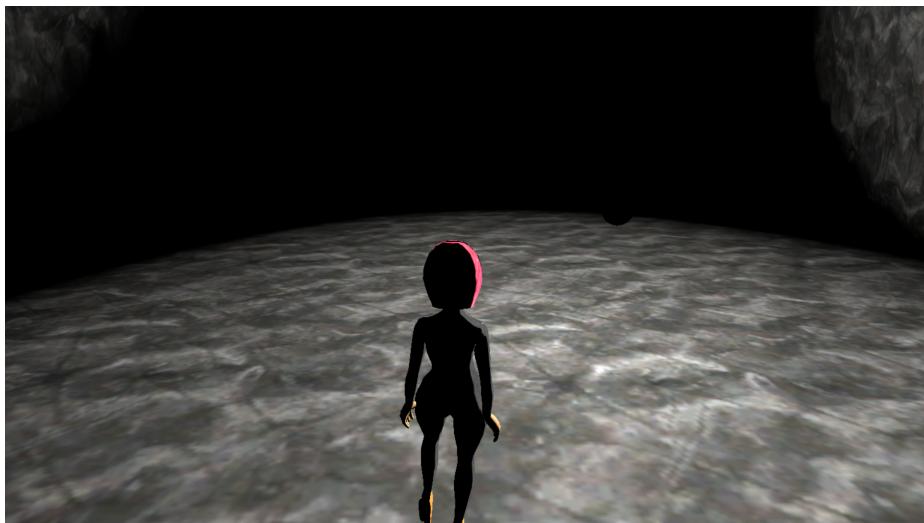
Sur une scène 3D, l'environnement de base est toujours le même : un ciel procédural bleu avec un sol noir, et des paramètres de luminosité par défaut, mais il est possible de la modifier grâce à un noeud "WorldEnvironnement". Le ciel étant à notre goût, il n'a pas été modifié. Cependant, le sol a lui été changé pour un bleu océan et la luminosité automatique désactivée. Lorsque l'on entre dans la grotte, la luminosité ambiante est désactivée pour qu'il y fasse vraiment très sombre, la seule source de lumière étant l'esprit.

Ci-contre, l'environnement normal puis l'environnement utilisé dans le jeu.



Et la différence entre la grotte sans et avec changement de la lumière ambiante :





3.3 Personnage (Jonathan P.)

3.3.1 Modèle

Le modèle 3D du personnage est le même depuis le début du projet. Il s'agit d'une jeune fille, qui à la base a une couleur uniforme blanchâtre. Au début, nous avons opté pour une texture de peau sur tout le corps (nous voulions faire un capuchon qui la recouvriraient) et roux pour les cheveux. les yeux sont tout noirs pour la scène finale. Lorsque nous nous sommes rendus compte que cette idée de cape n'était pas viable, nous avons opté pour une texture de tissus noir sur le corps et de cuir sur les chaussures.



3.3.2 Animations

Les animations ont été implémentées grâce à Mixamo, un site internet permettant de rigger (donner un "squelette" à un modèle dans le but de le faire se mouvoir). Le nombre et l'utilisation des animations sont passés par bon nombre d'états. Au début elles étaient au nombre de 9 : rester debout, marcher, courir, sauter, tomber, hocher la tête, remercier, rire et se relever. En avançant dans la conception, nous nous sommes rendus compte que certains n'avaient pas lieu d'être. Par exemple, sauter et tomber seraient utiles uniquement si des obstacles nécessitant de tels mouvements existaient, ce qui n'est pas le cas. Au final les 6 animations retenues sont :

- Rester debout, marcher et courir : les comportements de base d'un personnage de jeu vidéo
- Se relever et rire, respectivement pour la scène d'introduction et de conclusion du jeu (à noter que "se relever" se joue aussi à la fin d'une énigme)
- Hocher la tête : lorsque une action (téléportation ou énigme) est déclenchée

Les restrictions sont assez simples : on ne peut pas se déplacer pendant une cinématique, et quand le personnage est immobile, "rester debout" se joue automatiquement.

3.3.3 Caméra

Le principe de gestion de la caméra repose sur un noeud spatial (nommé target) placé environ au centre de la tête du personnage. Celui-ci sert de référentiel à la caméra qui est son fils dans l'arborescence, donc lorsque l'on applique une rotation au noeud target, la caméra tourne pour rester au même endroit dans son référentiel local, résultant en une rotation autour de la tête du personnage. Afin de calculer la rotation en question, nous récupérons les coordonnées (x,y) de la souris par rapport au centre du viewport, et on applique la rotation sur x au noeud target et celle en y au personnage, pour qu'il soit toujours droit à la caméra mais que celle-ci puisse tourner, puis on force la souris à se replacer au centre du viewport pour qu'il n'y ait pas de déplacement pendant que la souris ne bouge pas. Afin qu'elle ne passe pas à travers les murs mais qu'elle se rapproche lorsque le personnage est dos à un obstacle, la caméra est déclarée comme fils d'un objet godot qui remplit très bien cette tâche et qui a d'ailleurs principalement été créé pour les caméras à la 3e personne comme la nôtre, dont la gestion est soumise à beaucoup plus de contraintes que les caméras à la première personne ou même les caméras 2D. Cet objet s'appelle une "SpringArm", et son principe est le suivant : sa base est fixe et un rayon rectiligne en sort dans une direction choisie, avec une distance maximale. Tous les objets enfants sont placés au bout du rayon, et si quelque chose vient couper ce dernier, les enfants sont avancés au point où le rayon est coupé.

3.3.4 Esprit

Le personnage est en permanence suivi par un esprit qui l'aide tout au long de l'histoire. Il prend la forme d'une sphère lumineuse et flotte près du personnage. Il s'agit d'une sphère avec une texture de lave/feu et une source de lumière au milieu, ce qui permettra notamment d'illuminer les endroits sombres. Pour pouvoir suivre le personnage, ils sont tous les deux enfants d'une même scène, qui sera celle instanciée dans le jeu, afin qu'ils aient la même position dans l'espace mais qu'ils bougent indépendamment l'un de l'autre.

L'esprit se dirige vers "SpiritAim" qui est fixe par rapport au personnage et bouge en même temps que ce dernier, et qui est situé en haut, à sa droite. Sur les axes x et z (les horizontales) il se dirige donc vers la SpiritAim, et sur l'axe y (le verticale) il suit une sinusoïde.

Ces actions sont appelées à chaque frame, la variable time s'incrémentant donc à chaque frame 1. C'est pour cette raison que toutes les valeurs sont très réduites, afin de privilégier les micro déplacement et garder un effet le plus fluide possible (le déplacement divisé par 40 et le cosinus par 20).

3.3.5 Déplacements

Pour les déplacements, il faut prendre le vecteur allant de la caméra à la target, et lui retirer sa composante en y et on le normalise, ce qui donne un vecteur unitaire dans la direction où est tournée la caméra. Ensuite on applique ce vecteur en translation au personnage pour le faire bouger. Pour cela, on utilise la fonction MoveAndCollide(Vector3*delta) du noeud KinematicBody (noeud permettant de créer des objets répondant aux collisions et aux lois de la physique). Cette fonction permet une translation qui s'arrête lorsqu'il heurte certains objets comme les autres noeuds du type. Enfin, on lui applique une translation vers le bas par la même méthode pour lui faire subir la gravité KinematicBody, ou bien les RigitBody par exemple. Comme dit plus tôt, le personnage est instancié dans une scène intermédiaire avec l'esprit. A cause de cela, il se déplace sur la carte, mais seul sa position à l'intérieur de la scène intermédiaire change, cette dernière reste fixe à l'intérieur du jeu. Ce problème de "position relative" n'est pas grave en soi pour le joueur, mais il a demandé des étapes en plus pour certaines actions : la sauvegarde et la téléportation. Dans le premier cas un système de signaux en chaîne transmet cette position relative jusqu'à la scène "game" et permet de sauvegarder dans de bonnes conditions (on fait la somme de la position in game et de la position relative).

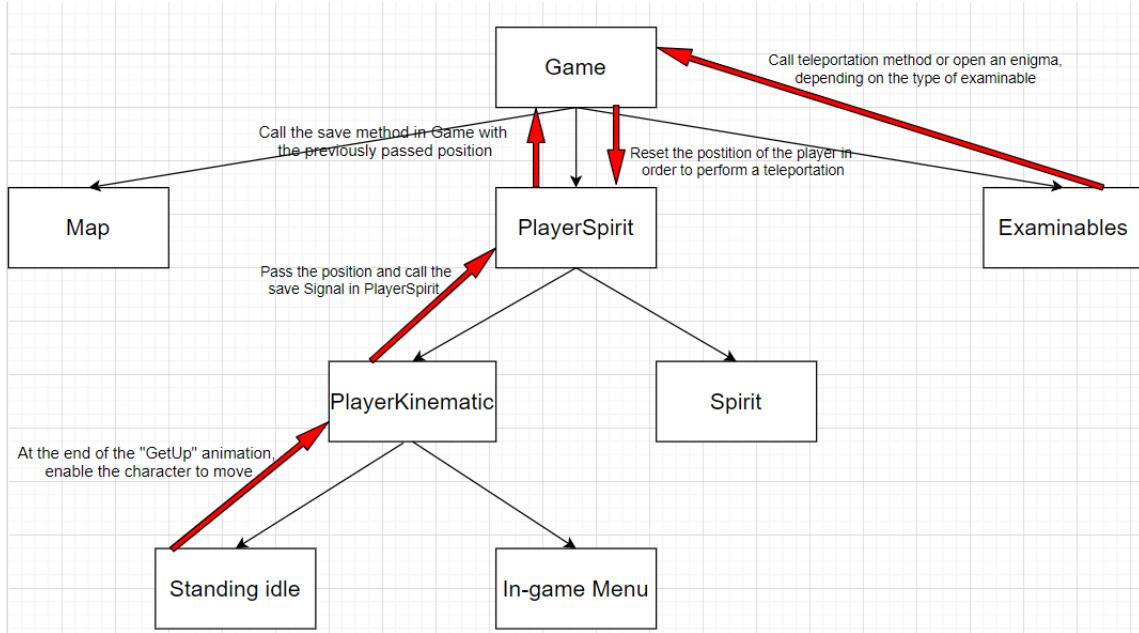
Dans le second cas il faut reset la position relative à (0,0,0), encore grâce à un signal, et ensuite changer la position de la scène intermédiaire.

Voici un schéma de la construction du projet.

Les rectangles représentent les différentes scènes importantes du jeu.

Les flèches noires montrent quelles scènes instancient quelles autres scènes.

Les flèches rouges représentent les signaux qui lient les différents scipts.



3.3.6 Cinématiques

Le jeu possède deux "cinématiques", dans les deux cas on prend le point de vue de l'esprit. La première se joue lorsque l'on arrive sur le jeu, l'esprit arrive de haut dans le ciel et se dirige à son emplacement normal tandis que le personnage se relève. Juste avant que l'on puisse voir son visage (un timer permet de connaître ce moment), la caméra change et devient celle qui reste dans tout le jeu, tandis que le personnage finit de se relever.

Dans la seconde, le personnage se met à rire de façon malsaine tandis que l'esprit se place devant elle, laissant enfin apparaître son visage, ses yeux noirs inhumains. Ensuite, il y a un gros plan sur son visage. Il s'agit du plot twist final du jeu.

3.3.7 Téléportation

Pour accéder à certains endroits de la carte, il faut téléporter le joueur :

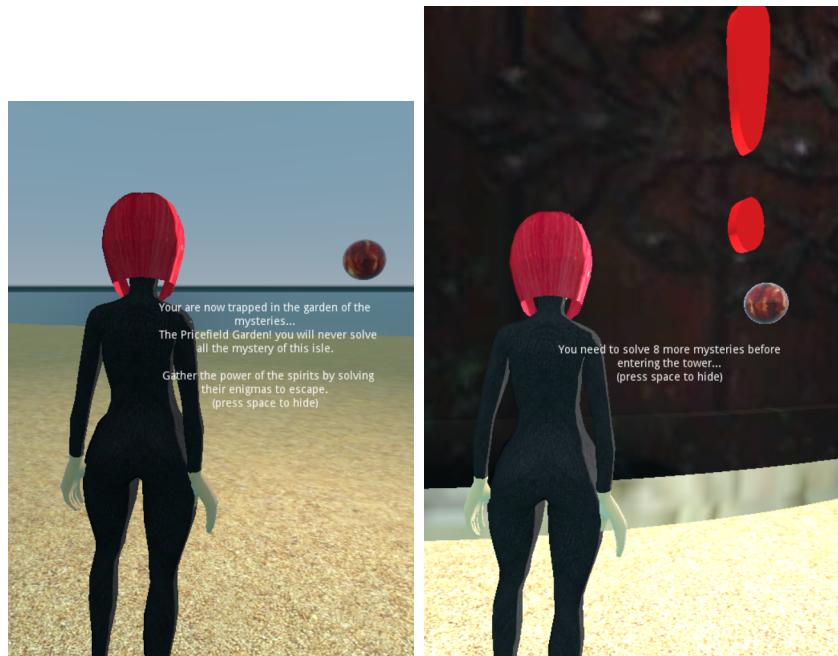
- Une des maisons du village : la porte n'étant pas animée, il faut le téléporter de l'autre côté.
- La grotte : n'étant pas reliée par un chemin à la carte, cela permet d'y accéder ou d'en sortir, en mettant à jour l'environnement (plus sombre) et la musique d'ambiance jouée.
- La tour : l'intérieur est plein (pas d'escaliers ou d'accès), mais il faut pouvoir accéder au sommet.

Pour téléporter le joueur, on ne peut pas changer directement sa position, il faut donc calculer la différence entre sa position actuelle et la nouvelle position voulue afin de trouver le vecteur selon lequel il faut le déplacer. Un signal est ensuite envoyé à **playerSpirit** pour reset la position relative du joueur. Pour lancer une téléportation, il faut effectuer une action sur un objet examinable.

3.3.8 HUD

Le personnage n'ayant pas d'item, de barre de vie ou autres informations d'habitude présentes dans le hud, il est la majorité du temps caché, il s'agit d'un popup contenant du texte qui se ferme avec, ici, la barre espace. Au début du jeu, une phrase expliquant le but du jeu apparaît, un autre lorsque l'on arrive au village afin de prévenir le joueur de bien l'explorer (si on le quitte il faut refaire le labyrinthe) et d'autres si l'on tente d'accéder à des parties de la carte non débloquées (passer le premier pont sans avoir réussi d'éénigme, monter dans la tour avec moins de 8 énigmes, ou accéder à la dernière île avec moins de 9 énigmes).

Voici deux exemples de textes :



3.4 Sauvegarde (Alexandre M.)

Nous avons dû créer une sauvegarde pour permettre au joueur de choisir s'il souhaite sauvegarder son avancée dans le jeu ou s'il veut créer une nouvelle partie. Pour cela nous avons créé plusieurs scripts C#.

Voici ci-dessous le script permettant de charger une partie sauvegardée au préalable. Ce script lit le fichier "save" pour redonner la position au personnage et les énigmes accomplies. Évidemment cela ne fonctionnera pas si le fichier save n'existe pas. En effet, il est impossible récupérer une partie inexistante.

```

void Load()
{
    try{
        using (StreamReader sr = new StreamReader("save"))
        {
            > var x = Convert.ToSingle(sr.ReadLine());
            > var y = Convert.ToSingle(sr.ReadLine());
            > var z = Convert.ToSingle(sr.ReadLine());
            > var enigmas = sr.ReadLine();

            > var pos = player.Translation;
            > var offset = new Vector3(x,y,z) - pos;
            > player.Translate(offset);

            > for (int i = 0; i < enigmas.Length; i++)
            {
                > if(enigmas[i] == '1')
                > {
                    > complete[i] = true;
                > }
            }
        }
        catch(Exception e)
        {

        }
    }
}

```

Contrairement à la fonction load, la fonction save a nécessité l'utilisation des signaux. Lorsque le joueur sélectionne le bouton "Save" du menu in-game, les coordonnées du personnage sont enregistrées et un signal est envoyé à un autre script.

```

case 1:
>     arrows.Translation = new Vector3(0, (float)0.2 ,0);
>     if (Input.IsActionJustPressed("ui_accept"))
>     {
>         var x = this.Translation.x;
>         var y = this.Translation.y;
>         var z = this.Translation.z;
>         EmitSignal("Save", x, y, z);
>     }
>     break;

```

Ce script récupère les coordonnées du personnage pour les envoyer au dernier script afin de suivre l'arborescence (présente dans la partie "3.3.5 Déplacements").

```

void _on_player_Save(float x, float y, float z)
{
    >     EmitSignal("Save", x, y, z);
}

```

Le script ci-dessous écrit donc la sauvegarde dans un nouveau fichier "save" écrasant le fichier existant. Il écrit dans ce fichier les coordonnées du personnage ainsi qu'une chaîne de caractères qui sert à connaître les énigmes résolues.

```
void _on_PlayerSpirit_Save(float x, float y, float z)
{
    using (StreamWriter sw = new StreamWriter("save")){
        x += player.Translation.x;
        y += player.Translation.y;
        z += player.Translation.z;
        string enigmas = "";
        foreach (var item in complete)
        {
            if (item)
            {
                enigmas += "1";
            }
            else
            {
                enigmas += "0";
            }
        }
        sw.WriteLine(x);
        sw.WriteLine(y);
        sw.WriteLine(z);
        sw.WriteLine(enigmas);
    }
}
```

3.5 Enigmes (Ilias B.)

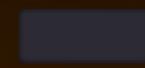
Avant de procéder à l'implémentation des énigmes, nous nous sommes demandés comment est-ce qu'elles fonctionneraient et seraient appelées. Comment nous allions placer les différentes informations sur la boîte à énigmes, comment est-ce qu'on allait recevoir les réponses du joueur etc... Et c'est à partir de cela que nous avons revu tout le système autour des énigmes que nous avions effectué la dernière fois. Le joueur sera donc confronté à deux types d'énigmes : des quizz et des QCM. Dans le premier cas, le joueur devra répondre en rentrant la réponse dans l'endroit prévu à cet effet.

Everyone on the hype train !

It takes 15 minutes to travel from Station A to Station B. It takes 5 minutes to travel from Station B to Station C. It takes 10 minutes to travel from Station C to Station D. However, it does not take 30 minutes to travel from Station A to Station D. Station A is the first station on the line. Station D is the end of the line. Since there are no switches on the line, how many minutes does it take to travel from Station A to Station D?

[Answer](#)

[Leave](#)



Dans l'autre, le joueur aura plusieurs choix et pourra cocher la/les réponses qu'il estime justes.

Rope Game

A farmer has a piece of land in a circle that he wants to divide into as many plots as possible to cultivate. To do this, he has five ropes at his disposal. Each rope must cross the circle in a straight line, but they can cross each other as many times as necessary. What is the largest number of plots he can obtain by dividing the land using the five ropes?

[Answer](#)

10

20

[Leave](#)

16

8



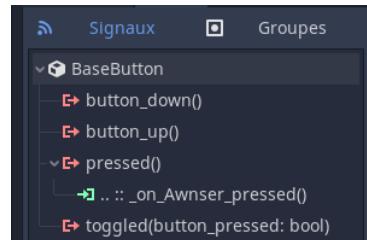
Pour accéder à une énigme, un signal permettant de récupérer la position relative² du joueur est envoyée et lance une sauvegarde normale. Nous ouvrons après cela la scène correspondant à l'énigme puis le joueur voit apparaître la boîte à énigme avec ses différentes spécifications (titre, texte mettant en avant la situation, un bouton pour quitter, un autre pour répondre et les possibilités de réponses).

```

1 reference
void enigmasSave()
{
    string x;
    string y;
    string z;
    string enigmas;
    using (StreamReader sr = new StreamReader("save"))
    {
        x = sr.ReadLine();
        y = sr.ReadLine();
        z = sr.ReadLine();
        enigmas = sr.ReadLine();
    }
    using (StreamWriter sw = new StreamWriter("save"))
    {
        sw.WriteLine(x);
        sw.WriteLine(y);
        sw.WriteLine(z);
        string tmp = "";
        for (int i = 0; i < enigmas.Length; i++)
        {
            if (i == EnigmeNumber-1)
            {
                tmp += '1';
            }
            else
            {
                tmp += enigmas[i];
            }
        }
        sw.WriteLine(tmp);
    }
}
0 references
private void _on_Awnser_pressed()
{
    if (!GetNode<CheckBox>("MCQ/Awnser_1").Pressed && GetNode<CheckBox>("MCQ/Awnser_2").Pressed &&
        !GetNode<CheckBox>("MCQ/Awnser_3").Pressed && !GetNode<CheckBox>("MCQ/Awnser_4").Pressed)
    {
        enigmasSave();
        GetTree().ChangeScene("res://PlayerTest.tscn");
    }
    else
    {
        GetNode<Popup>("Popup").PopupCentered();
    }
}

```

3



-
2. Voir : 3.3.4
 3. Exemple d'un QCM

Par la suite, nous avons dû mettre en place des noeuds qui vont récupérer les informations émises par le joueur lors de la résolution de l'énigme. Ils sont au nombre de deux et sont fondamentaux pour le code de cette partie du jeu.

Lorsque le joueur répond à l'énigme et clique sur le bouton "Answer", le noeud de l'image de dessus récupère la/les réponses sélectionnées et vérifie si la réponse donnée correspond à celle présente dans la fonction. A la suite de cela, si c'est juste, une sauvegarde du nombre d'énigmes résolues est faite puis le joueur se retrouve dans le monde extérieur de nouveau. Sinon, un popup apparaît pour notifier le joueur du fait qu'il n'a pas trouvé la bonne réponse et qu'il peut recommencer.



Quant à ce noeud, il récupère le clique du joueur sur le bouton "Leave", emmenant ainsi le joueur vers la carte.

Au départ, le joueur se retrouve directement confronté à une énigme (qui est facile cependant) afin de lui montrer dès le début les mécaniques de ce système d'énigmes. Par la suite, le joueur aura encore 9 énigmes à résoudre afin de connaître le secret caché du jeu. La difficulté de ces dernières est cependant croissante au fur et à mesure que le jeu avance. Il y a au total deux énigmes dans l'île de début, trois dans le village, trois dans la grotte, une dans la tour et une dans la zone de fin du jeu.

4 Site web (Ilias B. et Jonathan P.)

Le site internet est fonctionnel, il est hébergé sur Github Pages et dispose de plusieurs onglets. Il est de plus doté des éléments importants à son bon fonctionnement. Un menu latéral permet de naviguer entre les pages, qui contiennent un titre et un contenu en plus de ce menu.

- L'accueil : il contient l'artwork de base du jeu, celui qui a été créé au premier jour de ce projet. Evidemment, les idées ont changé depuis et le jeu sera sans doute différent de cette idée.
- Le développement : il contient les liens des rendus en PDF et du cahier des charges.
- La galerie : elle présente un pèle-mèle d'images prises tout au long du projet.
- Téléchargement : il contient le lien de téléchargement du jeu une fois celui-ci achevé.
- La soluce : il y a dans cette partie toutes les solutions des énigmes avec une explication lorsqu'elles sont compliqués.
- Les contacts : les visages, noms, et adresses mail de toute l'équipe.

5 Récit de la réalisation

Général

Suite à un départ assez lent dû à un manque d'organisation, alors que nous ne nous rendions pas compte de la charge de travail que représente un projet comme celui-ci, les choses ont peu à peu accélérées, ce qui ne nous a tout de même pas épargné le "rush de fin de projet" que l'on nous avait promis. Malgré cela, le projet est à une phase où il peut être considéré comme jouable, à défaut d'être fini à 100%. Nous sommes assez fiers du résultat, même s'il est un peu différent de ce que nous avions en tête. Les difficultés n'ont pas eu raison de la bonne entente au sein du groupe et l'entraide est restée un maître mot jusqu'à la fin.

Jonathan Poelger

Ce projet a été pour moi séparé en deux grandes parties bien distinctes. Au début et pendant le temps qui a passé jusqu'à la seconde soutenance, je me suis occupé presque exclusivement du personnage, qui à lui seul représente plus de "code" que tout le reste du jeu réunis. Au départ, lorsque j'ai émis le souhait de m'occuper de cette partie du projet, je ne m'attendais pas à ce que la tâche soit aussi complexe, et surtout aussi longue. Tout d'abord, la recherche du modèle 3D et des animations, choses que je n'avais jamais fait auparavant, ont demandés plus de recherches que de réelles réflexions, mais des problèmes de rendu et de compatibilité ont rendu la tâche intéressante car cela m'a permis de comprendre beaucoup de choses sur l'animation 3D. Et ensuite venu une multitude de petites parties, toujours avec leur lot de bugs en tout genres (par bug j'entends manque de rigueur ou inattentions de ma part). C'est comme cela que se sont peu à peu ajoutés la caméra, les déplacements, l'esprit et les collisions. Quand est venu l'heure de réaliser la cape, j'avais en tête une idée précise de ce que je voulais obtenir, mais rien n'a pu y faire. Après avoir écumé internet et testé tout ce qui me passait par la tête, nous en avons discuté avec tout le groupe et nous avons fini par admettre que cela n'était pas une partie "essentielle" du jeu et décidé de nous tourner vers une solution beaucoup plus simple pour les vêtements : des textures apposées à même le personnage. Passé ce cap qui aura retardé grandement le personnage, il a pu être fini avec les détails qui lui manquaient : l'ajout de gravité, les cinématiques, le HUD et la résolution de bugs restants concernant les animations et les déplacements. La seconde grande partie a été la création de la map. Nous pensions pouvoir tous y participer mais il était plus pratique qu'une partie du groupe s'en charge. Grâce aux modèles de Benoît, j'ai donc pu commencer à construire le monde qui allait servir de décor à notre jeu. Au début, la recherche de modèles, le texturing et la gestion des collisions étaient assez fastidieux mais peu à peu cela s'est accéléré et en relativement peu de jours (au rythme "rush projet") elle était finalisée. Il a ensuite fallu gérer l'accès aux différentes parties de la map tout en aidant à l'implémentation de la sauvegarde et des énigmes. Le jeu commençait à prendre sérieusement sa forme définitive et finalement une version suffisamment aboutie a pu voir le jour. En parallèle de ces tâches "personnelles", il a fallu faire le site internet, chose que j'avais déjà pu expérimenter en terminale. Au final, malgré les moments de frustration et de fatigue, avec un peu de recul, ce projet a été largement positif. Entre l'approche conséquente

de la 3D, la gestion d'un projet de relativement grande envergure, l'apprentissage de nouveaux outils informatiques et un semblant de management d'équipe, je pense que ce projet m'a beaucoup apporté, d'autant que le résultat est à la hauteur de ce que nous avions espéré.

Alexandre Martinez

Quand j'ai commencé à créer le menu principal de notre jeu je trouvais cela intéressant. Je n'avais jamais utilisé Godot auparavant mais après avoir créé la première version du menu je me suis vite rendu compte des possibilités qu'offre Godot surtout maintenant que nous pouvons écrire des scripts en C#. Je suis passé par plusieurs manières différentes pour créer ce menu et l'utilisation des signaux est également très pratique pour relier tous les éléments différents de notre projet. De plus, avec le menu principal, il y avait la sauvegarde du jeu. Heureusement que j'ai pu compter sur l'aide de mes coéquipiers dans cette tâche pour pouvoir les aider en retour sur d'autres parties du projet. Je dirais presque "heureusement" qu'il y a eu des bugs durant l'avancée du projet d'une part car un projet parfait me paraît louche et d'autre part car il y avait toujours des problèmes à régler et des coéquipier à aider. Je ne me suis pas du tout ennuyé en faisant ce projet et j'espère avoir d'autres occasions de travailler avec la même équipe. Malgré les problèmes de connexion internet que j'ai pu rencontrer à cause de cette période difficile qu'est le confinement, je ne me suis pas découragé et j'ai continué à travailler sur mes tâches grâce à l'aide des autres membres du groupe. Ce projet m'a aidé à travailler en équipe et à en apprendre beaucoup sur Godot et autres logiciels utilisés.

Ilias Belkhder

J'avais assez hâte quant à la création de ce jeu vidéo et dans l'idée de voir ce que cela donnerait au final. En effet, je voyais cela comme un projet de très grande envergure, ce qui était une grande première pour moi. Ce projet s'est avéré difficile mais je le termine avec un sentiment positif. En effet, cela a été l'occasion pour moi d'en apprendre beaucoup sur le travail d'équipe car il a fallu s'adapter à de nombreuses reprises. J'ai dû apprendre à travailler avec des personnes que je connaissais en ami seulement et pas au niveau de la rigueur de travail. Cela ne m'a cependant pas dérangé. Je suis fier que notre groupe ait réussi à produire un jeu fonctionnel qui nous plaît. J'ai également pu mettre en œuvre les différentes connaissances en programmation que j'avais pu acquérir avec l'école dans un projet concret de plus grande envergure que les travaux pratiques hebdomadaires. Ce projet m'a permis de m'entraîner sur un aspect qui me semble fondamental dans le développement qui est le passage de l'idée au code. En effet, c'était surtout le cas pour les énigmes. J'ai tenté, comme dit dans ma partie au sujet des énigmes, plusieurs codes qui n'ont pas marché ou je ne voyais pas comment finir. Par exemple, dans le code finale, pour que le joueur fasse une énigme, nous ouvrons carrément une nouvelle scène et c'est pour cela qu'à la fin de chaque énigme, le jeu doit se recharger entièrement de nouveau. Au début, j'avais fait une liste de tuples dans laquelle j'allais mettre toutes mes énigmes puis ensuite j'allais, dans la même classe, donner les informations de ces dernières (titre, texte, etc.). Sauf qu'après, j'ai remarqué que ça allait me prendre trop de

temps à essayer de trouver comment faire la suite. J'ai donc changé de méthode à deux reprises avant de me mettre d'accord avec Jonathan sur la méthode actuelle.

Cette dernière période est malgré tout arrivée très vite. En effet, seul un mois séparait le dernier rendu de celui-ci. Cependant, nous avons tout de même pu nous accrocher. Nous avons réussi à finir ce qui avait été démarré et ce qui était toujours en développement lors de la deuxième soutenance.

Benoît Brice

Au commencement, j'ai eu beaucoup de mal à débuter le projet car c'était la première fois que je faisais un projet comme celui-ci. De plus, j'étais content d'avoir cette occasion de pouvoir créer un jeu vidéo, tout en ayant le choix par rapport au thème de celui-ci. Cependant, je me suis presque indirectement découragé car je pensais que j'étais incapable de créer un jeu en 3D, dont je pensais que l'élaboration était bien plus compliquée. Heureusement, le fait d'avoir observé sur certains réseaux tels que Youtube ou Discord la création de certains jeux vidéo qui étaient partagés par des gens passionnés m'a permis de me lancer et de tester pas mal de choses. J'ai d'ailleurs eu rapidement l'envie de créer une map en 3D car la forme, la couleur et l'ambiance que dégagent ces maps étaient pour moi ce qui me faisait apprécier un jeu ou non. Cependant, malgré mes recherches, je n'ai pas trouvé de moyen de créer cette map et j'ai donc passé énormément de temps à chercher comment faire. Le problème étant que peu de personnes ayant utilisé Godot ont fait un jeu en 3D. Ce qui fait qu'au début nous n'avions que très peu d'informations. Malgré cela, un soir, une tendance sur Youtube m'avait montré une personne utilisant un logiciel qui permettait de créer un terrain pour divers usages ainsi que pour un jeu vidéo. J'ai donc commencé à explorer toutes les possibilités ainsi que les outils mis à disposition par le logiciel et j'ai pu concevoir le terrain que je souhaitais un mois plus tard. Par la suite, le texturing et l'ajout d'objets et de décors dans la map sont des choses que j'ai vraiment pris beaucoup de plaisir à faire, me permettant justement de faire partager l'ambiance et les sensations que nous voulions donner au joueur. Finalement, ce projet que je craignais beaucoup au départ m'a permis de me rendre compte que le travail en équipe est quelque chose qui peut beaucoup apporter car nous nous sommes fait confiance tout au long du projet, amenant ainsi du soutien et de l'entraide. Tout cela nous a ensuite permis d'atteindre nos objectifs. Je suis donc au final très content d'avoir pu participer à un projet tel que celui-ci.

6 Conclusion

Ce projet est arrivé à son état final, ce qui n'était pas gagné d'avance. Toutes les parties essentielles sont achevées, et ce, même si certains points ne sont pas tout à fait fonctionnels, l'ensemble reste cohérent et jouable.

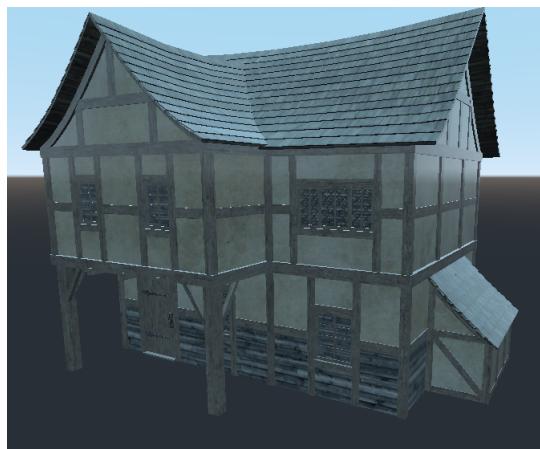
Nous sommes heureux d'avoir finalement puachever ce projet que nous avons imaginé il y a des mois de cela. Nous étions au départ un groupe d'amis qui partageaient des hobbies. Nous sommes ensuite devenus une équipe qui arrive à s'entraider et à se motiver pour arriver à faire des choses qui nous dépassent.

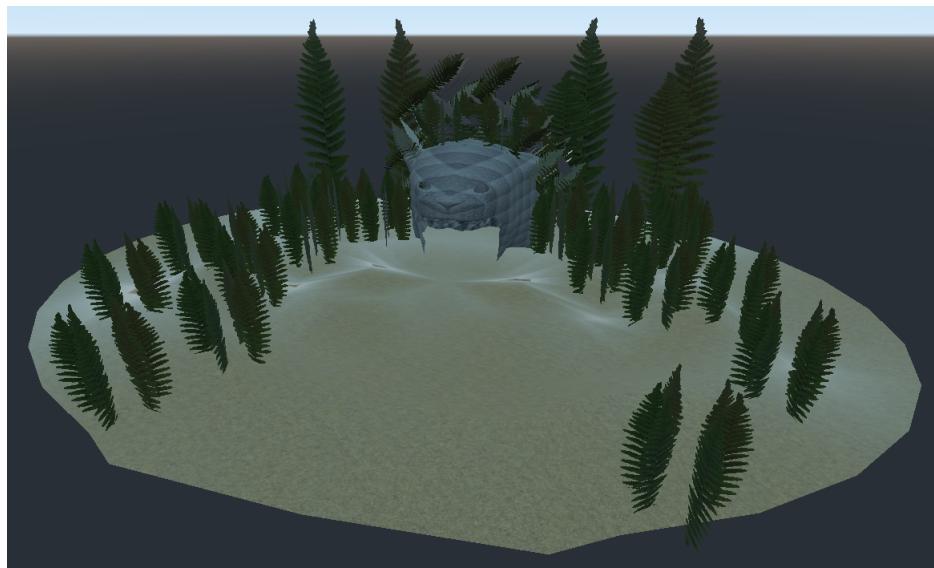
Malgré un départ assez lent compte tenu du manque d'organisation et de la mauvaise façon dont nous avions appréhendé ce travail (comme à la première soutenance), nous nous sommes rattrapés pour le final et nous sommes aujourd'hui très fiers du résultat. Bien que la recherche de modèle 3D ait pris du temps étant donné la complexité de trouver de bons modèles gratuits, cela nous a permis de donner notre touche personnelle sur ce projet, que nous avons pu créer de A à Z.

Ce projet aura aussi été pour nous l'occasion d'apprendre à travailler efficacement en tant qu'équipe, mais aussi d'améliorer nos connaissances dans différents domaines de programmation et de les utiliser dans la conception d'un projet de longue haleine. Le domaine de la conception de jeux vidéo nous a motivé et, pour un premier projet, nous pensons avoir atteint les objectifs de départ. Cela a été une expérience très intéressante et enrichissante pour nous.

7 Annexes

Modèles 3D du jeu :





Sites Utilisés :

- ShareCG (Modèles 3D)
- Free 3D (Modèles 3D)
- Turbosquid (Modelès 3D)
- 3DTextures (textures)
- Textures.com (textures)
- GodotEngine (Documentation godot)
- docMicrosoft (Documentation c)
- Reddit (forums informations)
- Stack Overflow (forums informations)
- Github pages and Jekyll (Upload du site)

Logiciels utilisés :

- Godot (moteur du jeu)
- Blender (3D et textures)
- Photoshop (GFX : images et retouches)
- Adobe Illustrator (GFX : images et retouches)
- WorlMachin (création 3D)
- Discord (Server Godot)