

# Rapport de Projet

*Akari*

Mai 2023

Groupe 2 - NAYRU



LÉPINE Marc-Aurèle (Chef de projet)

MARRASSÉ Damien

SANTACREU Alexandre

ANDRE Luca

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Rappel du projet . . . . .	4
<b>2</b>	<b>L'équipe NAYRU et le projet Akari</b>	<b>5</b>
2.1	Présentation de l'équipe NAYRU . . . . .	5
2.2	Origine et nature du projet Akari . . . . .	6
2.2.1	Origine du projet . . . . .	6
2.2.2	Nature du projet . . . . .	7
2.3	Objet d'étude du projet . . . . .	8
2.3.1	Intérêt pour le joueur . . . . .	8
2.3.2	Intérêt pour le groupe . . . . .	8
2.4	État de l'art . . . . .	9
2.4.1	L'isométrique . . . . .	9
2.4.2	Le Voxel art . . . . .	9
2.4.3	Les jeux d'énigmes . . . . .	10
2.4.4	Les jeux d'actions . . . . .	11
<b>3</b>	<b>Découpage du projet</b>	<b>12</b>
3.1	Description des tâches . . . . .	12
3.1.1	Modélisation 3D . . . . .	12
3.1.2	Sons . . . . .	12
3.1.3	Animations . . . . .	12
3.1.4	Texturing . . . . .	12
3.1.5	Diagramme d'interactions . . . . .	12
3.1.6	Intégration des mécaniques . . . . .	12
3.1.7	Intelligence artificielle . . . . .	12
3.1.8	Réseau . . . . .	12
3.1.9	Intégration du design . . . . .	12
3.1.10	Site Web . . . . .	13
3.2	Répartition des tâches . . . . .	13
3.3	Avancement du projet . . . . .	14
3.4	Outils et logiciels utilisés . . . . .	15
3.4.1	Unity . . . . .	15
3.4.2	Blockbench . . . . .	15
<b>4</b>	<b>Retour sur la première soutenance</b>	<b>16</b>
4.1	Modèles . . . . .	16
4.2	Création des maps . . . . .	19
4.3	Caméra . . . . .	20
4.4	Déplacement du joueur . . . . .	21
4.5	Chargement des levels . . . . .	23
4.6	Site Web . . . . .	24
4.7	Avancement du projet . . . . .	25

<b>5 Retour sur la deuxième soutenance</b>	<b>26</b>
5.1 Structuration du code . . . . .	26
5.2 Entités . . . . .	29
5.2.1 Squelette . . . . .	29
5.2.2 StoneGuardian . . . . .	31
5.3 UI . . . . .	35
5.3.1 Transitions . . . . .	35
5.3.2 Menus . . . . .	37
5.4 Multijoueur . . . . .	38
5.5 Site Web . . . . .	38
5.6 Avancement du projet . . . . .	39
<b>6 État final du projet Akari</b>	<b>40</b>
6.1 Entités . . . . .	40
6.1.1 StoneGuardian . . . . .	40
6.1.2 SandGuardian . . . . .	41
6.1.3 StoryTeller . . . . .	42
6.1.4 Panneaux . . . . .	42
6.2 Maps . . . . .	43
6.2.1 Plaines . . . . .	43
6.2.2 Désert . . . . .	44
6.3 Énigmes . . . . .	45
6.3.1 Levier . . . . .	45
6.3.2 Récompense . . . . .	45
6.4 Gameplay . . . . .	46
6.4.1 Combats . . . . .	46
6.4.2 Mort . . . . .	46
6.5 Sauvegarde . . . . .	47
6.6 Avancement du projet . . . . .	48
<b>7 Synthèses personnelles</b>	<b>49</b>
7.1 Marc-Aurèle LÉPINE . . . . .	49
7.2 Damien MARRASSÉ . . . . .	49
7.3 Alexandre SANTACREU . . . . .	50
7.4 Luca ANDRE . . . . .	50
<b>8 Conclusion</b>	<b>51</b>

# 1 Introduction

## 1.1 Rappel du projet

Notre projet est un jeu d'action-aventure mêlant combat et énigmes. Il se décompose en deux phases de gameplay. La première dans laquelle le joueur doit libérer du mal les zones dans lesquelles il s'aventure. Nous nous sommes principalement inspirés du gameplay de *The Legend Of Zelda : Twilight Princess* pour cette phase, en ce sens que le joueur est éprouvé par chaque boss à la fin de chaque zone. Puis, dans une seconde phase, le joueur peut choisir de résoudre des énigmes afin d'obtenir des améliorations qui lui faciliteront l'aventure. Cette phase est laissée au choix du joueur. S'en passer peut être une façon de rendre le jeu plus difficile.

Pour cette dernière soutenance, nous avons terminé toutes les tâches qu'ils nous manquaient pour aligner notre jeu à notre vision de celui-ci.

## 2 L'équipe NAYRU et le projet Akari

### 2.1 Présentation de l'équipe NAYRU

Le groupe NAYRU se compose de quatre membres :

#### **LEPINE Marc-Aurèle**

Ferru d'informatique depuis tout jeune, j'ai toujours voulu créer des jeux vidéos. Je trouve que cette forme d'art permet de faire passer énormément d'idées tout en laissant au joueur la liberté d'imaginer et de découvrir, d'être curieux. J'attends beaucoup de ce premier gros projet, notamment une certaine rigueur dans la création, mais surtout le plaisir de la création et de pouvoir partager quelque chose qui me plaît avec notre groupe.

#### **SANTACREU Alexandre**

Je suis Alexandre Santacreu, passionné des sciences et plus particulièrement de leurs applications numériques. La programmation est un de mes sujets favoris et mettre en œuvre mes connaissances et en engranger de nouvelles par l'intermédiaire de ce projet me semble être un engagement bénéfique dans l'avancement de mes études. J'ai tout simplement hâte de commencer et mener à bien ce projet avec l'ensemble de notre groupe.

#### **MARRASSÉ Damien**

Je m'appelle Damien MARRASSÉ et je suis étudiant à EPITA après avoir fait une terminale spécialités Mathématiques et NSI. Depuis tout jeune intéressé par l'informatique, j'ai hâte de faire ce projet de groupe qui nous permettra d'apprendre à gérer la création et la mise en œuvre d'un projet.

#### **ANDRE Luca**

Je m'appelle Luca ANDRE, actuellement étudiant Epita, j'envisage de travailler dans la cyber-sécurité après mes études. Ayant fait les spécialités Mathématiques et NSI, ce projet de groupe me paraît très intéressant et très enrichissant pour pouvoir acquérir de nouvelles compétences.

## 2.2 Origine et nature du projet Akari

### 2.2.1 Origine du projet

Notre équipe à peine créée, nous nous sommes rapidement mis d'accord pour développer un jeu vidéo. En premier lieu, nous avons réfléchi au **type de jeu** que nous voulions faire, nous cherchions à réaliser quelque chose d'intéressant et d'original qui lierait action et réflexion. Un subtile mélange entre un jeu d'énigme à la Capitaine Toad, et de combat en temps réel à la Zelda Twilight Princess. Nous avons donc tout simplement décidé de mélanger ces deux genres là.

La deuxième étape de notre réflexion était de convenir du **style graphique** que nous aimions adopter. Nous voulions presque à l'unanimité quelque chose de simple à créer mais qui permettrait l'ajout de détails. C'est ainsi que nous est venu l'idée du Voxel Art. Un style graphique plaisant à l'oeil qui rappelle le pixel art, qui, à l'aide d'un jeu de lumières, lui ajoute de la profondeur.

Enfin, la dernière étape était de convenir de **l'objectif du joueur**. Nous ne voulions pas axer notre jeu autour d'une quelconque histoire, néanmoins nous étions tous d'accord sur le fait qu'un objectif de jeu était primordiale pour rendre tout jeu intéressant.

### 2.2.2 Nature du projet

Notre jeu s'articulera autour de deux phases de gameplay, en alternant logiquement suivant l'histoire du jeu. Le premier concept nous est venu du jeu *The Legend of Zelda : Twilight Princess* dans lequel est opposé deux mondes "rivaux", celui de la Lumière, et celui des Ombres. L'élément perturbateur est une volonté du monde des Ombres de prendre le contrôle de celui de la lumière. Ainsi, et de la même manière, nous souhaitons faire en sorte que le joueur puisse alterner entre deux mondes, l'un dont la libération représente l'objectif principal, et l'autre, plus tranquille, dans lequel le joueur serait libre d'aller et venir comme il souhaite; l'intérêt résidant en la résolution d'éénigmes. Le second concept nous est venu du jeu *Captain Toad : Treasure Tracker* et correspond au fait de pouvoir jouer sur l'angle de vue afin de résoudre certaines éénigmes permettant d'avancer (bien que dans notre jeu, nous ayons décidé de les rendre optionnelles).

Pour faire simple, nous aurions :

1. **Du combat** contre des monstres et des boss dans les zones soumises aux ténèbres. C'est ce qui constituera l'objectif principal du joueur : ramener la lumière sur l'entièreté de son univers.
2. **Des éénigmes** faisant usage du point de vue et de l'angle de la caméra. Elles auront pour but de permettre au joueur d'avoir un réel sentiment d'amélioration, mais aussi de donner la possibilité aux plus téméraires de les passer pour rendre le jeu plus difficile à souhait.

## 2.3 Objet d'étude du projet

### 2.3.1 Intérêt pour le joueur

Le développement d'un tel jeu est très intéressant d'un point de vue connaissances personnelles car il fait aussi bien appel à **l'imagination**, à la **modélisation**, et à la **conception** qu'à la programmation en elle-même. De plus, il est nécessaire de connaître les outils que l'on utilise. Ainsi cela nous apprends à **chercher**, **se renseigner**, **essayer**, et voir les possibilités qui nous sont offertes aussi bien que les limites, permettant leur anticipation.

### 2.3.2 Intérêt pour le groupe

Du point de vue du travail d'équipe, ce projet permet d'instaurer un véritable environnement professionnel. Il permet de bien cerner la nécessité de **structurer** le projet par le présent cahier des charges notamment, afin que chaque membre de l'équipe ait une idée claire des tâches qui lui sont données, ainsi que de l'avancement technique du projet dans sa globalité.

## 2.4 État de l'art

### 2.4.1 L'isométrique

Le mot "isométrique" vient du latin *-iso* (égal) et *-métrique* (qui peut être mesuré). Il s'agit d'un type de perspective dans laquelle les trois côtés d'un cube sont représentés avec la même importance à l'écran.

Initialement, ce style de rendu graphique était utilisé pour **compenser les faibles capacités des anciennes consoles** qui ne pouvaient pas faire de rendus 3D.

Le premier jeu ayant utilisé cette technique est *Knight Lore*, développé en 1984 par le studio *Rare* anciennement *Ultimate Play The Game*. Dans celui-ci, le protagoniste doit amener à un magicien au centre du labyrinthe des ingrédients qui permettront de lever la malédiction qui a transformé le joueur en loup garou.

Depuis, le genre s'est démocratisé, et avec l'arrivée de la 3D, de nombreux développeurs ont eu l'idée de croiser les genres. Réalisant ainsi des jeux tels que *Crossy Road*, ou encore *Roller Coaster Tycoon*.

### 2.4.2 Le Voxel art

Un Voxel est un néologisme créé à partir du mot *pixel* et *volume*. Il signifie littéralement pixel en volume. Le voxel art est une discipline qui consiste en la création de modèles 3D composés de plus ou moins de voxels, représentant la plus petite unité mesurable de l'objet rendu. Il est directement issu du Pixel Art, qui consiste en la création d'image 2D, à cela qu'il rajoute une dimension de dessin supplémentaire, il permet aussi de nombreux détails tels que par exemple un jeu d'ombres et de lumières plus important dûs à la dimension supplémentaire.

### 2.4.3 Les jeux d'énigmes

Le genre énigme ou de réflexion est un genre qui soumet le joueur à des mécanismes dont il doit comprendre le fonctionnement afin de réaliser une certaine action lui permettant de gagner.

Nous pouvons citer d'anciens jeux issus de ce genre tels que:

1. *Tetris*, créé en 1984, par *Aleksei Pajitnov*. Dans ce jeu, des formes géométriques variées tombent du haut de l'écran vers une plateforme en bas sur laquelle elles vont reposer. Le but du joueur est de faire en sorte que la pile créée ne remonte pas jusqu'en haut de l'écran. Le joueur doit donc faire en sorte d'orienter les pièces de façon à ce qu'elles remplissent une rangée, ce qui aura pour effet de les faire disparaître.
2. *FEZ*, plus récent, est un jeu vidéo développé en 2012 par *Polytron Corporation*. Dans celui-ci, le joueur devra récolter des morceaux d'un cube dispersé dans le monde afin de rétablir l'ordre de l'univers. Au premiers abords, ce jeu semble être fait en 2D. Mais son essence se dévoile lorsque le joueur obtient la capacité de faire tourner le monde entier vers la droite où la gauche. Cette mécanique est au cœur de toutes les énigmes de FEZ.
3. *Captain Toad : Treasure Tracker* dans lequel le joueur incarne Toad, - un personnage de la licence Mario Bros - qui à pour objectif de sauver Toadette. Pour cela, il devra traverser chaque monde avant de passer au suivant. La mécanique principale de ce jeu est que Toad ne peut pas sauter. De ce fait, le joueur doit contrôler l'orientation du monde dans lequel évolue Toad afin de lui permettre de résoudre les énigmes et d'avancer.

#### 2.4.4 Les jeux d'actions

Les jeux d'actions forment une très grande partie, si ce n'est la majorité, des jeux présents sur le marché.

Il est à noter que ce genre de jeu se déroule généralement en suivant un même schéma narratif, à savoir :

1. **L'intrigue**, ou la situation initiale, qui sert de point de départ et d'entrée en matière pour le joueur. Elle permet de comprendre l'univers, de poser le contexte, et servira l'implication du joueur.
2. **L'élément perturbateur** qui vient cour-circuiter le déroulement de l'intrigue. Cette étape est primordiale car elle permet de définir (de manière sous-entendue ou explicite) les objectifs du joueur. (Par exemple, sauver le royaume d'Hyrule lorsque celui-ci se fait envahir par des monstres du Crénuscle ).
3. **Les péripéties** qui, dans le cadre d'un jeu vidéo, ne sont généralement pas ou très peu représentées, mis à part la stricte avancée du joueur et son passage par certains points clés du déroulement de l'histoire. (Par exemple l'obtention d'un pouvoir supplémentaire, d'un quart de cœur en plus, ou plus simplement une cinématique peuvent servir à souligner ce point).
4. **Le dénouement**. Il s'agit la plupart du temps de la fin du scénario. Cela peut être le boss final, ou alors la toute dernière quête. L'idée ici est que le spectateur puisse répondre à la question : "Le héros va-t-il réussir ?".
5. **La situation finale**. Il peut s'agir d'une cinématique, d'une toute dernière phase de gameplay, mais elle doit conclure l'histoire en annonçant les conséquences de la victoire ou de la défaite du personnage principal face à l'élément perturbateur.

## 3 Découpage du projet

### 3.1 Description des tâches

#### 3.1.1 Modélisation 3D

La création de maps sur laquelle le joueur évolura, modélisation des boss, du joueur, des objets en général.

#### 3.1.2 Sons

Création des sons, tels que les bruitages ou les musiques.

#### 3.1.3 Animations

Animation des objets du jeu, bouclantes et non-bouclantes.

#### 3.1.4 Texturing

Coloration des objets et création des materials.

#### 3.1.5 Diagramme d'interactions

Création du diagramme d'interaction entre tous les objets.

#### 3.1.6 Intégration des mécaniques

Programmation des mécaniques de jeu.

#### 3.1.7 Intelligence artificielle

Gestion de l'IA des ennemis

#### 3.1.8 Réseau

Création du NetworkManager, gestion du multijoueur.

#### 3.1.9 Intégration du design

Mise en relation des parties programmatives et du design. (Tel que l'adaptation des objets à la lumière ambiante par exemple).

### 3.1.10 Site Web

Création et hébergement du site web.

## 3.2 Répartition des tâches

Nous avons choisi de nous répartir en deux sous-groupes de travail. Un groupe chargé de la programmation logique, et un groupe chargé du rendu général du jeu.

	Marc-Aurèle	Alexandre	Damien	Luca
Modélisation 3D			Assistant	Responsable
Sons			Responsable	Assistant
Animations			Responsable	Assistant
Textures			Assistant	Responsable
Diagramme d'interactions	Responsable	Assistant		
Développement des mécaniques	Responsable	Assistant		
Intelligence Artificielle	Assistant	Responsable		
Réseau	Responsable	Assistant		
Site web			Assistant	Responsable
Intégration du design	Assistant	Responsable		

### 3.3 Avancement du projet

Voici à quoi ressemble l'avancement de notre projet depuis le début.

## Tableau d'avancement

	Soutenance n°1	Soutenance n°2	Soutenance n°3
Intelligence Artificielle	10%	60%	100%
Multijoueur	10%	75%	100%
Modèles, Textures & Animations	40%	75%	100%
Level design & Maps	25%	45%	100%
Gameplay : Combat	10%	50%	100%
Gameplay : Énigmes	0%	10%	100%
Site web	75%	95%	100%

## 3.4 Outils et logiciels utilisés

Nous n'avons pas utilisé beaucoup de logiciels, car ceux qu'on a utilisés nous ont apporté tout ce dont on avait besoin.

### 3.4.1 Unity

Nous avons utilisé Unity pour former notre jeu. C'est **LE** logiciel que nous avons principalement utilisé pour le code, rassembler les modèles et créer notre jeu en général. On avait déjà des bases avant de commencer le projet, mais nous avons aussi appris de nouvelles fonctionnalités qu'offre Unity qui nous a beaucoup aidés.

### 3.4.2 Blockbench

Blockbench est le logiciel que nous avons utilisé pour faire tous nos modèles, textures et animations. Il possède de nombreuses fonctionnalités qui nous ont permis de réaliser des modèles efficacement car ce logiciel est utilisé pour *Minecraft* et ses mods, qui possède un style graphique similaire à notre jeu. En revanche, les tutoriels liés à ce logiciel sont presque toujours tournés pour *Minecraft*, ce qui nous a parfois ralenti dans la recherche d'une fonctionnalité.

## 4 Retour sur la première soutenance

Pour la première soutenance, nous avons implémenté les bases de notre jeu comme la caméra, les déplacements ou encore le chargement des levels. Il nous était important d'avoir des bases solides et d'avoir réglé tous les problèmes que nous aurions pu rencontrer avant d'implémenter des choses plus compliquées.

### 4.1 Modèles

Nous avons réalisé une grande partie des modèles qui composent le terrain de la première zone. Ces models sont fait de telle sorte à pouvoir être déclinés en plusieurs version différentes selon la zone dans laquelle ils seront utilisés.



Figure 1: Dark Matter



Figure 4: Ground Block



Figure 7: Path Block - 3 Ends



Figure 2: Grass Block 1



Figure 5: Path Block - 1 End



Figure 8: Path Block - 4 Ends



Figure 3: Grass Block 2



Figure 6: Path Block - 2 Ends



Figure 9: Path Block - Center



Figure 10: Path Block - Border



Figure 11: Path Block - Corner



Figure 12: Path Block - Line

De même, nous avons modelisé quelques détails, à savoir des **herbes**, des **rochers**, des **buissons** et une **barrière**. Ces détails servent principalement à décorer le terrain.



Figure 13: Grass tips 1



Figure 16: Rock 1



Figure 19: Bush 1



Figure 14: Grass tips 2



Figure 17: Rock 2



Figure 20: Bush 2

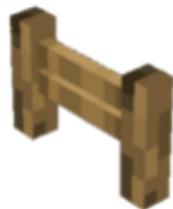


Figure 15: Fence



Figure 18: Rock 3



Figure 21: Bush 3

## 4.2 Creation des maps

Pour integrer nos models dans unity et pouvoir les utiliser, nous avons commence par setup un repository github separe de celui du jeu. Ce faisant, nous pouvons travailler simultanement sur le jeu tout en permettant aux autres membres de lquipe de pouvoir ajouter de nouveaux models.

Lorsqu'un modele est termin (modlis textur), nous l'avons export au format **.obj**, qui est directement lisible par Unity. Nous avons donc import le **modle**, la **texture**, et nous avons cre un **material** propre au modle.

Vu que notre rendu est de type **voxel art**, nous avons chang le **mode de compression** par dfaut de chaque texture pour "None", ainsi que le **Filter Mode** `a "Point : No Filter". Cela a pour effet de dtailler chaque pixel au lieu d'un rendu flou.

Lorsqu'un modele est prt `a tre utilis, nous lui ajoutons un composant **Box Collider** pour qu'il ragisse aux collisions. Cela tant dit, certains modles tels que les herbes ou les petits rochers n'en ont pas, et cela afin de ne pas gner le dplacement du joueur inutilement.

Lorsque tous les composants ncessaire ont t ajouts `a notre modle, nous en crons un **prefab** afin de le rutiliser facilement sans avoir `a refaire toute les tapes prcdentes.

### 4.3 Caméra

Pour implémenter la caméra, nous avons suivi notre plan initial qui était de donner un rendu isométrique. Néanmoins, nous avons été surpris de constater qu'une caméra placée à équidistance de notre personnage selon les 3 axes X, Y, Z étaient relativement jolie. Nous avons donc décidé de conserver cet angle pour voir ce que cela donnerait avec l'évolution du jeu.

Nous avons donc commencé par donner à la caméra un script qui lui permet de suivre le joueur en tout instant à travers l'appel natif de la méthode ***LateUpdate()***. Nous verrons plus loin pourquoi nous n'utilisons pas ***Update()*** directement dans la section 2.4.

```
private void LateUpdate()
{
    Vector3 finalposition = target.position + offset;
    Vector3 smoothed = Vector3.Lerp(transform.position, finalposition, smoothness * Time.deltaTime);
    transform.position = smoothed;

    transform.LookAt(target);
}
```

Figure 26: Script de la caméra

Nous commençons par calculer la position **relative au joueur de la caméra** (*target* étant le composant ***Transform*** du joueur, et *offset* un ***Vector3*** qui décale la caméra), puis nous utilisons la fonction statique ***Lerp()*** (pour linear interpolation) afin de créer une multitude de positions successive que va prendre la caméra en partant de sa propre position (***transform.position***) jusqu'à la position voulue (***finalposition***) avec un écart entre chaque point à valeur de ***smoothness***. Ce dernier champ étant public, il nous donne le contrôle depuis l'éditeur sur la vitesse à laquelle la caméra atteindra la position finale.

Enfin nous terminons par donner à chaque frame sa nouvelle position à la caméra, et finissons par un appel à la méthode ***LookAt(Vector3)*** nous permettant de toujours avoir le joueur au centre de la caméra.

## 4.4 Déplacement du joueur

Avant toute chose, il faut noter que le personnage dispose d'un composant ***Rigidbody*** en plus du ***BoxCollider*** de chaque objet. Ceci aura son importance pour la suite.

Nous avions initialement décidé de faire déplacer le joueur selon les axes X et Z du repère par défaut proposé par unity. Mais nous avons rapidement constaté que cela ne faisait pas vraiment "naturel". De ce fait, nous avons plutôt opté pour un déplacement suivant **l'axe de la caméra**. Pour ce

```
void Update()
{
    Vector3 vel = rb.velocity;

    Transform t = singleCam.transform;
    Vector3 forward = t.forward;
    Vector3 side = t.right;
```

Figure 27: Script du joueur - 1/4

faire, nous avons du commencé par récupérer les *vecteurs directionnels* de notre future "repère" depuis notre caméra passé en paramètre. (***forward*** et ***right***). Puis nous supprimons la composante en Y des vecteurs obtenus

```
forward.Set(forward.x, newY: 0, forward.z);
forward *= Input.GetAxis("Vertical") * speed;

side.Set(side.x, newY: 0, side.z);
side *= Input.GetAxis("Horizontal") * speed;
```

Figure 28: Script du joueur - 2/4

afin de récupérer le plan dans lequel le joueur se déplacera. De plus, nous augmentons la taille de ces vecteurs par la vitesse du joueur et changeons leur sens à l'aide des appels ***Input.GetAxis("Horizontal" ou "Vertical")*** pour couvrir les 4 direction de déplacement horizontales (***devant***, ***derrière*** couverts pas le vecteur ***forward***; et ***gauche***, ***droite*** couverts par le vecteur ***side***). Puis finalement, nous assigneons une composition des deux vecteurs

```

    vel.x = forward.x + side.x;
    UpdateJump(vel.y);
    vel.z = forward.z + side.z;

    rb.velocity = vel;

```

Figure 29: Script du joueur - 3/4

à chacun des axes du vecteur ***velocity*** du rigidbody de notre joueur. Notons ici que nous ne modifions pas directement la position du joueur, mais plutôt la vitesse de son rigidbody. Cela à pour conséquence de laisser Unity se charger du changement de position tout en calculant correctement les collisions potentielles.

Comme nous pouvons le voir sur la Figure 27, nous calculons la saut dans une fonction à part, par souci de clareté du code. Cette fonction nous

```

void UpdateJump(float yComp)
{
    isGrounded = yComp <= jumpThreshold && yComp >= -jumpThreshold;

    if (Input.GetButtonDown("Jump") && isGrounded)
        rb.AddForce(new Vector3(0, yJumpForce, 0), ForceMode.Impulse);
}

```

Figure 30: Script du joueur - 4/4

permet de vérifier si le joueur est **immobile sur l'axe Y** avant d'appliquer une **force d'impulsion** sur son rigidbody (toujours par souci de calcul de collision). AUssi, nous voyons que nous utilisons un **seuil** qui nous permet de vérifier que le joueur soit immobile au sol et non pas dans les air. En effet, nous avons passé une variable ***jumpThreshold*** assez petite telle que d'une frame à l'autre dans les airs, le joueur ne soit pas considéré comme "au sol". Le seul moyen pour que cette condition soit vérifiée est qu'il soit au sol depuis plusieurs frames.

## 4.5 Chargement des levels

Notre jeu se découpe en plusieurs petits niveaux qui se chargent lorsque l'on franchit un certain endroit de chacune des maps. Pour pouvoir charger nos scènes les unes depuis les autres, il nous fallait trouver un déclencheur. Nous avons donc choisi d'utiliser des ***GameObjects*** vides ne possédant qu'un ***BoxCollider*** passé en mode ***Trigger***. Premièrement, cela permet

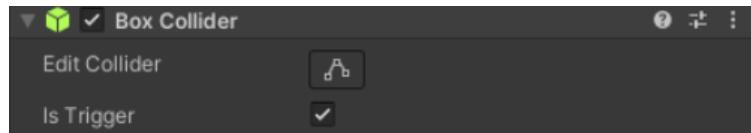


Figure 31: Trigger

au joueur de désactiver la collision, mais cela permet aussi d'utiliser la paire d'événements : ***OnTriggerEnter(...)*** et ***OnTriggerExit(...)***. Comme

```
private void OnTriggerEnter(Collider collider)
{
    GameObject gameObject = collider.gameObject;

    if (gameObject.CompareTag("Player")) { // Seul le player peut charger une zone
        LevelLoader.LoadScene(toScene);
    }
}
```

Figure 32: OnTriggerEnter

```
public static void LoadScene(string scene)
{
    SceneManager.LoadScene(scene);
}
```

Figure 33: Loader

nous le voyons sur ces deux images, nous récupérons l'objet à l'origine de la détection, puis nous vérifions qu'il s'agisse bien du joueur. Pour cela, nous avons attribué un tag "Player" au joueur. Puis nous déclenchons le chargement de la scène dont le nom est passé en argument. L'utilisation

d'une classe statique nous permet de donner à chaque "Triggerer" (cf. objet vide servant uniquement de détecteur). le même script en changeant juste la scène d'arrivée.

## 4.6 Site Web

Nous avons commencé à créer le site internet qui servira de plateforme pour le lancement et la promotion de notre jeu. Nous avons déjà créé le squelette de toutes les pages du site. Cela signifie que nous avons défini **l'architecture de notre site web** en y incluant les **différentes sections, les menus et les fonctionnalités nécessaires**.

Maintenant, nous travaillons sur le thème du site pour qu'il soit en rapport avec notre jeu vidéo. Nous voulons que le site **reflète l'univers et l'atmosphère du jeu**. Nous devons donc travailler sur **les titres, les descriptions et les images** pour donner aux visiteurs une idée claire de ce qu'ils peuvent attendre du jeu.

En plus de cela, nous sommes en train de mettre en place un moyen de téléchargement du jeu sur le site. Cela permettra aux joueurs de télécharger facilement le jeu à partir du site web, une fois qu'il sera disponible.

Nous travaillons également sur la présentation de notre équipe et des logiciels que nous utilisons pour créer le jeu. Nous voulons que les visiteurs puissent avoir une idée de qui nous sommes et de la façon dont nous travaillons pour créer ce jeu.

## 4.7 Avancement du projet

### Tableau d'avancement

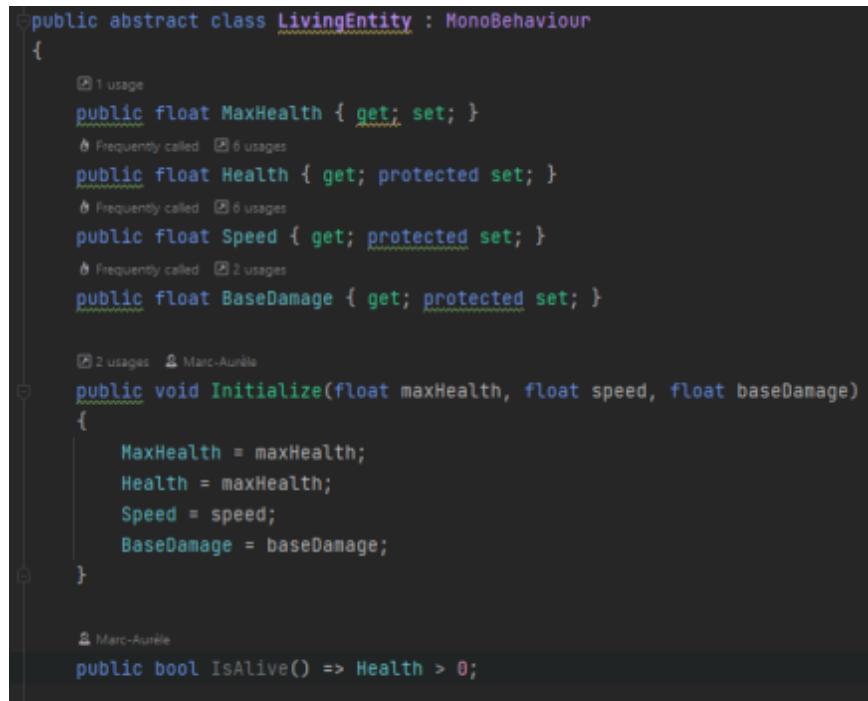
	Soutenance n°1
Intelligence Artificielle	10%
Multijoueur	10%
Modèles, Textures & Animations	40%
Level design & Maps	25%
Gameplay : Combat	10%
Gameplay : Énigmes	0%
Site web	75%

## 5 Retour sur la deuxième soutenance

### 5.1 Structuration du code

Comme tout bon développeur objet, nous avons cherché à généraliser un maximum de comportements de nos ennemis dans une class nommée **LivingEntity** qui comme son nom l'indique représente "ce qui est vivant".

Cette dernière étend directement de **MonoBehaviour**, permettant ainsi d'être passée en component à un **GameObject**. Néanmoins, nous l'avons dite **abstraite** car nous ne voulions pas pouvoir l'instancier directement. (Ce qui est logique finalement car il n'existe pas d'entités qui est "juste" une entité. Elle doit forcément être quelque chose pour exister en jeu, avoir une essence comme diraient les philosophes. Par exemple être un *Joueur*, un *Squelette*, un *Poulet*, etc...)



```

public abstract class LivingEntity : MonoBehaviour
{
    #region Properties
    public float MaxHealth { get; set; }
    public float Health { get; protected set; }
    public float Speed { get; protected set; }
    public float BaseDamage { get; protected set; }
    #endregion

    #region Constructors
    public void Initialize(float maxHealth, float speed, float baseDamage)
    {
        MaxHealth = maxHealth;
        Health = maxHealth;
        Speed = speed;
        BaseDamage = baseDamage;
    }
    #endregion

    #region Methods
    public bool IsAlive() => Health > 0;
    #endregion
}

```

Figure 1: Classe LivingEntity

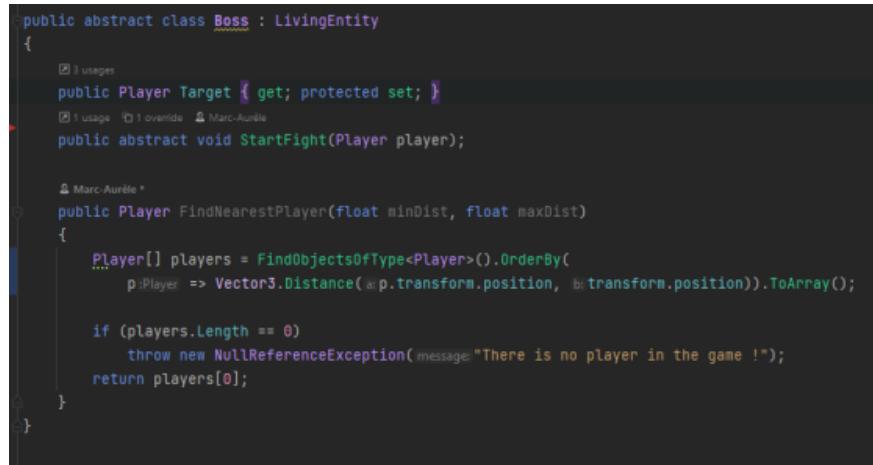
Comme nous pouvons le voir sur la Figure 1, les attributs **MaxHealth**, **Health**, **Speed**, et **BaseDamage** sont tous ceux que nous avons choisi de généraliser pour l'instant.

De plus, nous avons créé la méthode **Initialize(...)** qui sert de constructeur alternatif qui pourra être override dans de potentielles classes filles si besoin.

Il est intéressant de noter que nous ne pouvons pas réellement utiliser de constructeur lorsqu'un objet étend de **MonoBehaviour** car cette classe appelle elle-même le constructeur lorsque l'objet est instancié en jeu. A la place, la documentation préconise d'utiliser **Awake()** ou **Start()** respectivement appelées lors de l'initialisation et de la première frame du jeu pour éviter tout conflit.

En somme, nous avons réinventé le constructeur, sauf que celui-ci est appelé dans la méthode Awake de nos classes filles.

De la même façon, notre gardien de pierre est une *LivingEntity*, mais il s'agit aussi d'un boss. Ainsi, nous avons décidé de créer la classe abstraite **Boss** qui agit comme une couche d'abstraction supplémentaire afin d'alléger encore plus le code de nos classes.



```

public abstract class Boss : LivingEntity
{
    public Player Target { get; protected set; }

    public abstract void StartFight(Player player);

    public Player FindNearestPlayer(float minDist, float maxDist)
    {
        Player[] players = FindObjectsOfType<Player>().OrderBy(
            p => Vector3.Distance(p.transform.position, transform.position)).ToArray();

        if (players.Length == 0)
            throw new NullReferenceException(message: "There is no player in the game !");
        return players[0];
    }
}

```

Figure 2: Classe Boss

Nous avions préalablement décidé que nos boss allaient ”s’activer” uniquement sous certaines conditions. A l’inverse des squelettes qui, eux, courrent vers le joueur dès qu’ils le voient; d’où la présence de la méthode abstraite **StartFight** qui prend en paramètre le joueur qui est à l’origine du lancement du combat.

L'attribut **Target** quant à lui détermine la cible actuelle du boss. Cette cible est susceptible de changer, notamment lorsque le joueur s'éloigne en multijoueur par exemple. D'où la présence de la méthode **FindNearest-Player** qui, assez explicitement, recherche le joueur le plus proche dans la scène courante, à l'aide de la fonction **FindObjectOfType()** fournie par UnityEngine.

De même, nous avons fait le choix de créer l'interface **IDamageable** qui contient une méthode **Hurt()** ainsi qu'une méthode **Kill()**.

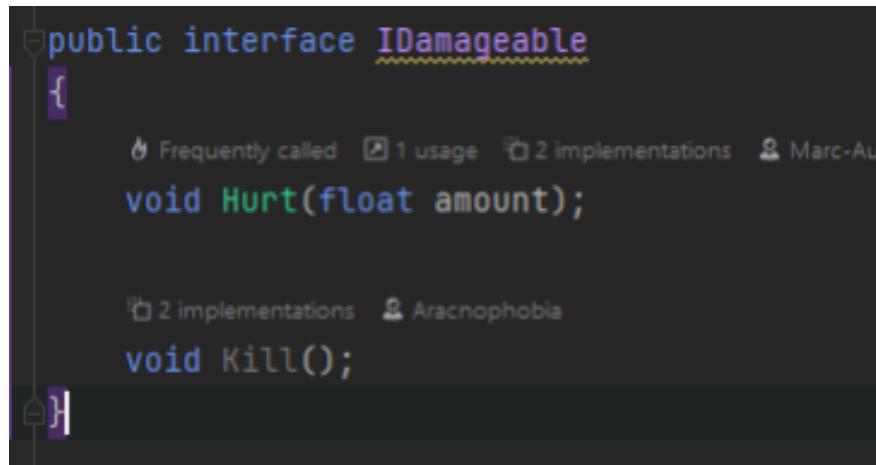


Figure 3: Interface **IDamageable**

Nous aurions aussi très bien pu implémenter directement ces méthodes en tant que méthode abstraite dans notre classe *LivingEntity* c'est vrai, mais en réalité, cela nous obligeait à faire en sorte que tous les monstres "vivants" puissent être tués.

Or, nous allons le voir plus loin, mais nous avions prévu un combat particulier pour notre boss qui nous empêchait de faire ce raccourci.

## 5.2 Entités

Pour la deuxième soutenance, nous avons terminé les modèles du **Squelette** ainsi que celui du **Gardien de pierre**.

### 5.2.1 Squelette

Le squelette est un monstre basique que nous pouvons mettre dans toutes les zones sans les dénaturer.

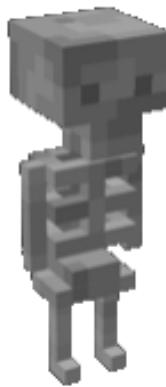


Figure 4: Squelette

Pour ce qui est des animations de notre squelette, nous avons réalisé une animation de **Marche**, de **Course**, de **Frappe**, de **Dégât**, et de **Mort**. L'animation de mort nous plaît beaucoup car nous voulions pouvoir garder

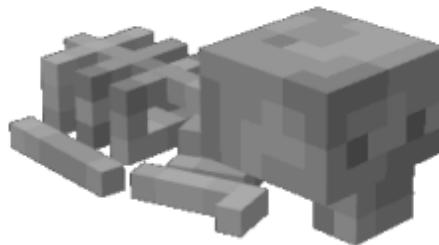


Figure 6: Squelette Mort

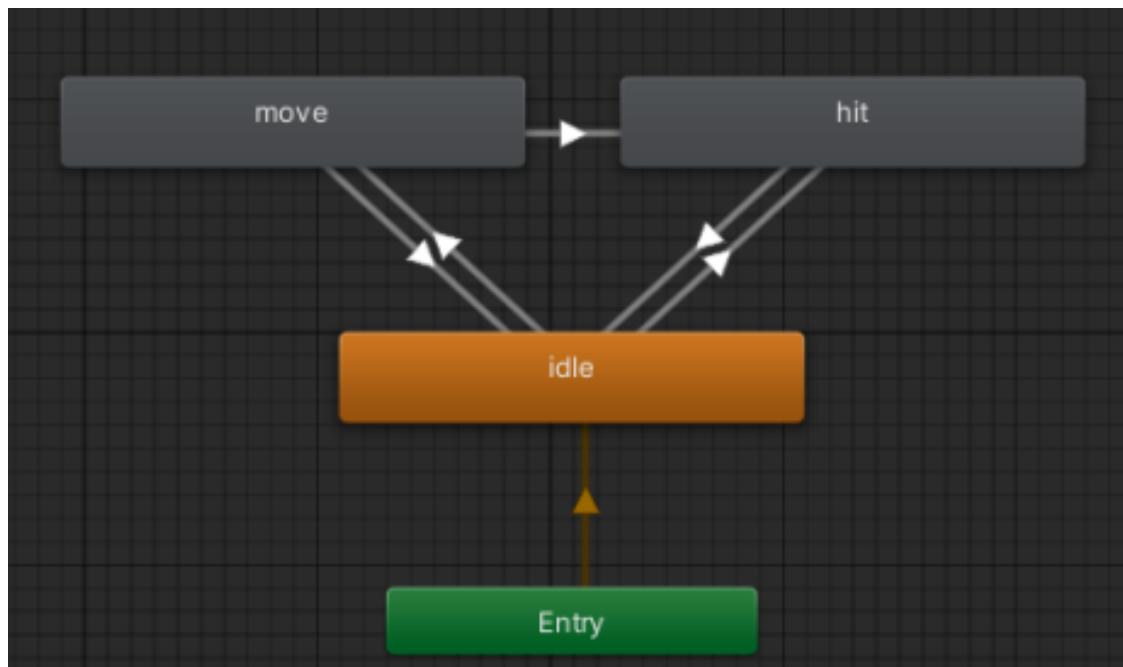
une trace visuelle des ennemis qu'aurait vaincu le joueur. Ici, la tête et les os du squelette resteraient par exemple au sol, là où fut tué le monstre.

Pour implémenter le squelette en jeu, on utilise un animator. Le squelette se comporte de cette façon : lorsqu'il est trop éloigné de son adversaire, il s'en approche et lorsqu'il est à la bonne distance il se met à l'attaquer. Il y

a donc plusieurs transitions possibles :

- il se trouve trop loin et se met à marcher pour se rapprocher (idle to move)
- il est assez proche et se met à attaquer le joueur (move to hit)
- il est déjà assez proche et peut attaquer le joueur (idle to hit) Cette situation

est donc modélisée par cet animator et les conditions sont chacune vérifiées dans des scripts liés aux différents états du squelette.



### 5.2.2 StoneGuardian

Pour le StoneGuardian, sa texture évoque la zone dans laquelle il se trouve. Par exemple, il a des l'herbe sur lui lorsqu'il est dans les plaines.

Aussi, nous avons choisi de faire un modèle qui soit en plusieurs morceaux. Tout d'abord car cela évoquait une sorte de force magique qui entretenait les pierres entre-elles, ce qui collait plutôt bien avec l'aspect "magie noire malfaisante" de notre jeu, mais aussi et surtout car cela s'avérait plus simple à animer.



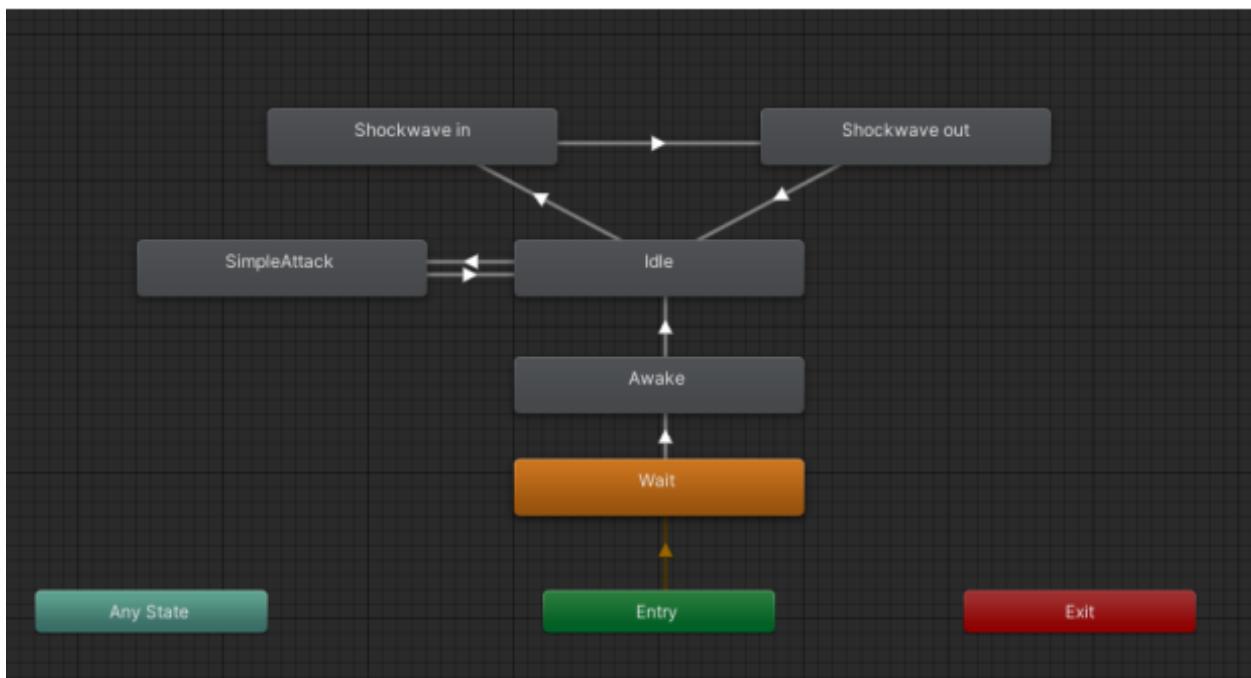
Figure 5: Gardien de pierre

Nous avons apporté une attention toute particulière aux animations de notre gardien de pierre car il s'agit d'un point clé de notre jeu, un des boss. Nous lui avons créé une animation de **Pré-Combat**, une d'**Eveil** (se joue lorsque je joueur entre dans la zone de trigger du combat), une d'**Idle**, une première pour une **Attaque simple**, une seconde pour une **Attaque puissante**, et enfin une animation de **Mort**



Figure 7: StoneGuardian avant le combat

L'animator de notre StoneGuardian nous permet d'agencer les différentes animations précédemment vues selon certaines conditions que nous décidons. Par exemple, ci-dessous, nous faisons en sorte que l'état *Awake* (état pendant lequel le gardien joue son animation d'éveil) ne se déclenche **uniquement lorsque le joueur entre en collision avec le déclencheur du combat**. De même, l'état *SimpleAttack* ne s'active qu'à un certain moment, lorsque l'IA du boss l'a décidé



Le combat se déroule en **deux phases**. La première dans laquelle le boss va lancer **une ou plusieurs attaques**, phase pendant laquelle le joueur ne peut pas attaquer le boss, puis dans un second temps, le boss va lancer une attaque suite à laquelle il va se retrouver au sol les bras détachés du corps. C'est seulement à cet instant que le boss est vulnérable et peut être attaqué. Lorsque cela se produit, le cœur du gardien devient visible et peut subir des dégâts.

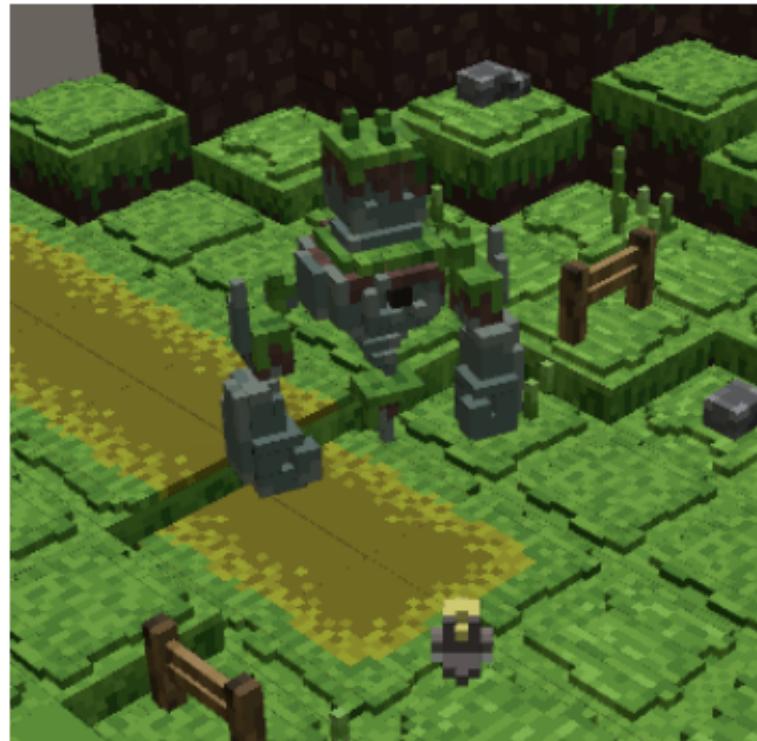


Figure 12: Phase invulnérable (interlude entre deux attaques)



Figure 13: Phase vulnérable

## 5.3 UI

### 5.3.1 Transitions

Nous voulons véritablement faire ressentir un changement lors que le joueur entre dans une zone de ténèbre. C'est donc pour celà que nous avons décidé de mettre en place une animation de transition.

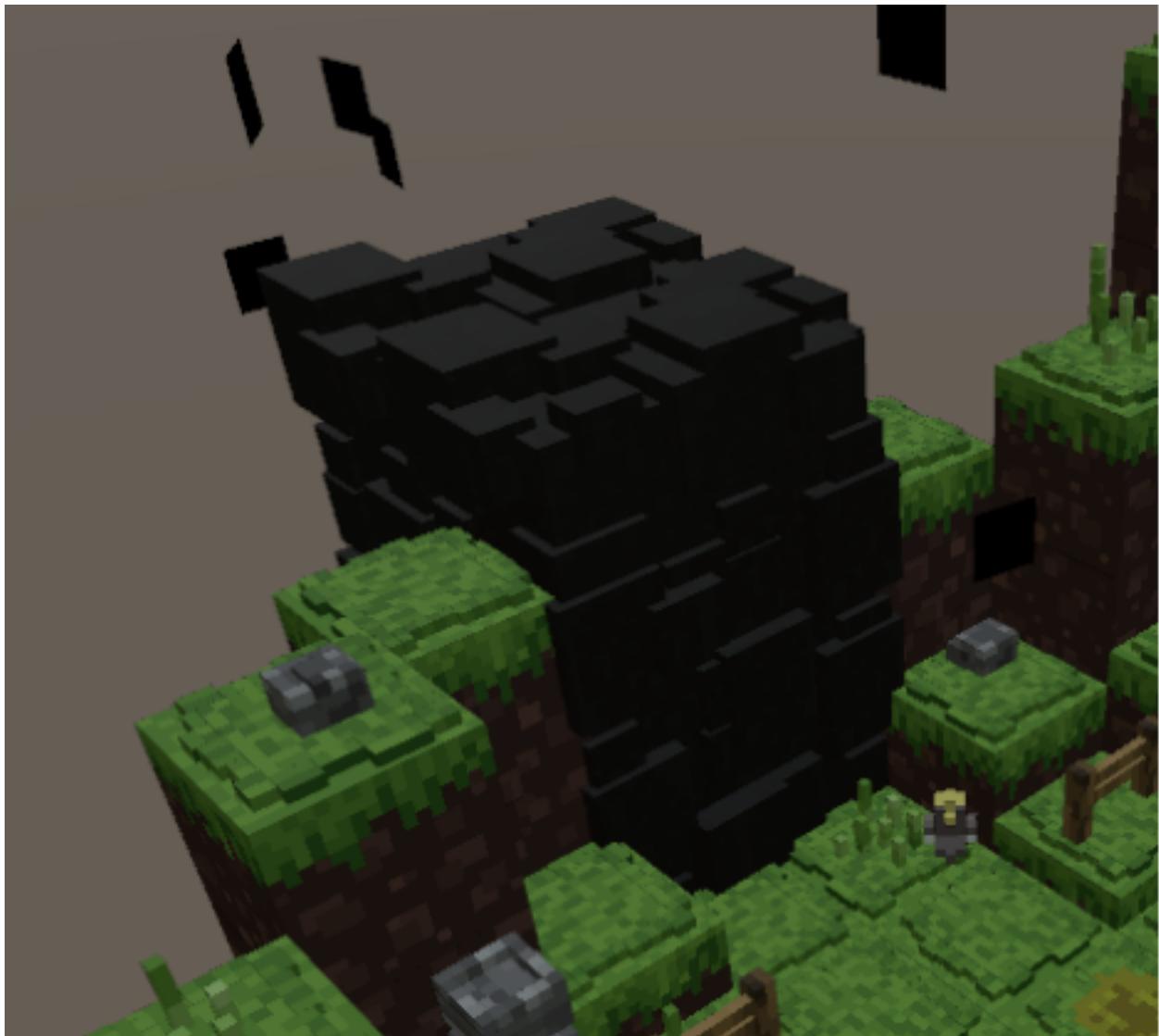


Figure 21: Entrée de la zone

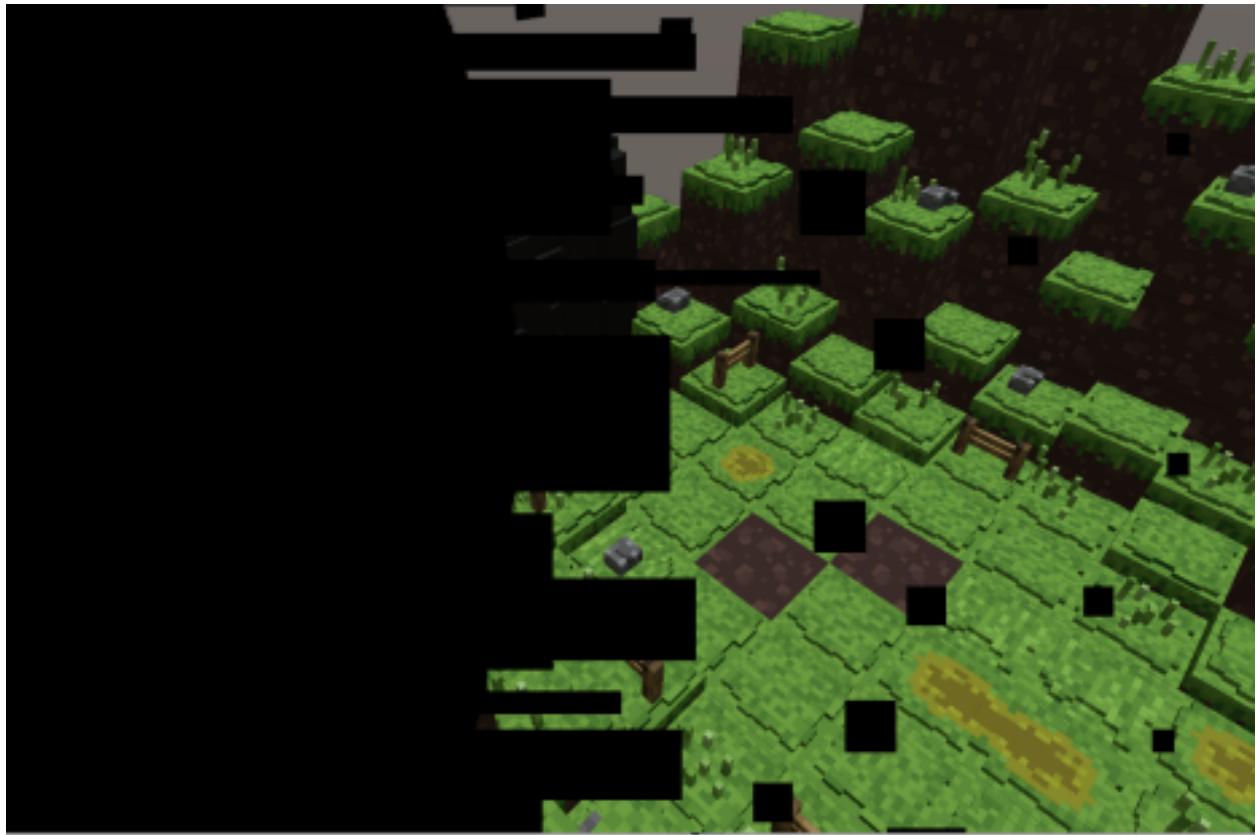


Figure 22: Transition (sortante)

L'animator va lancer une animation *In* puis attendre que l'animation d'entrée se termine pour charger la prohaine zone. Puis il envoie une animation *Out* et attends qu'elle se termine pour laisser le joueur continuer.

### 5.3.2 Menus

Nous avons aussi réalisé plusieurs menus afin de faciliter la navigation et le choix du mode de jeu.

Nos menus se décomposent de la façon suivante :

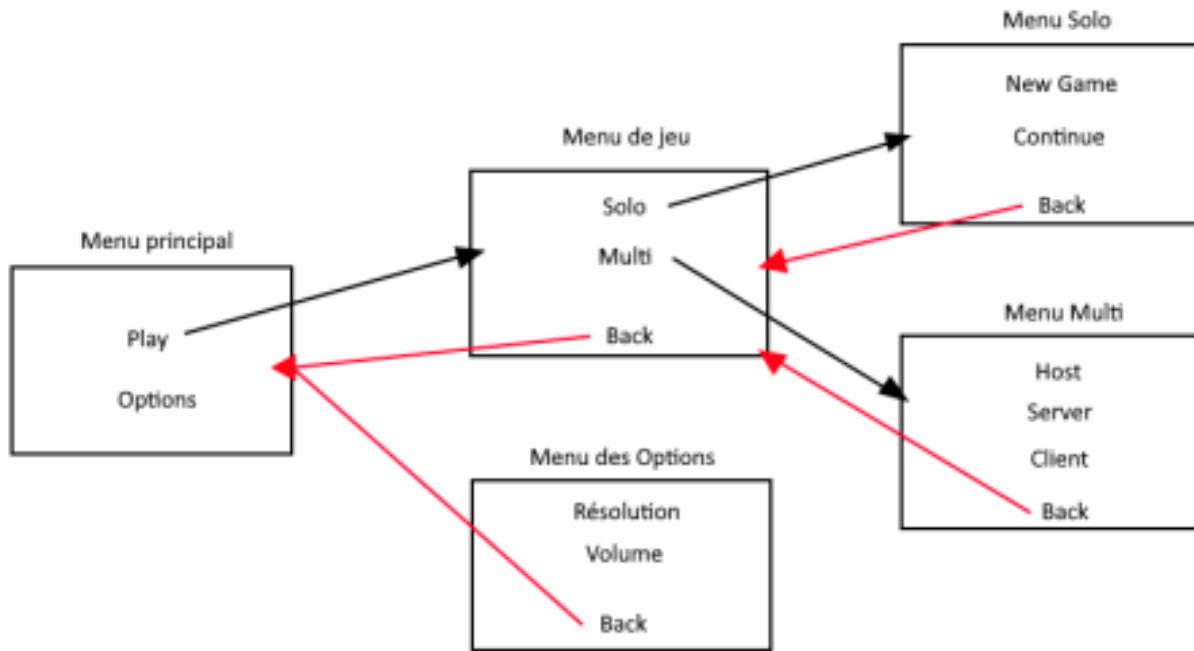


Figure 24: Diagramme d'intéraction des menus

Pour réaliser ceci, nous nous sommes grandement inspiré du système de *Finite State Machine* que nous avions déjà utilisé pour notre boss mais en le modifiant un peu. En effet, les menus s'y prêtent plutôt bien car on ne peut être que sur un seul menu à la fois, et on ne peut aller que vers un seul autre menu.

## 5.4 Multijoueur

Pour le mode multijoueur, nous avons décidé de ne pas utiliser la bibliothèque *Mirror*, mais plutôt d'utiliser **NetCode for GameObjects**. Notamment car Netcode proposait une classe *NetworkBehaviour* qui hérite directement de *MonoBehaviour*, nous permettant ainsi de ne pas avoir à re-coder la majorité de notre code et de pouvoir le réutiliser. Aussi, nous avons mis en place un système en LAN.

Nous avons donc créé une classe *ServerPlayer* qui hérite de *NetworkBehaviour* qui, après attribution à un *GameObject*, est utilisée par le *NetworkManager* pour créer les joueurs lorsqu'ils se connectent.

De plus, nous avons créé la scène BossRush qui sera celle dans laquelle les joueurs arriveront lors de leur connexion; la même dans laquelle le boss rush se déroulera.

Ainsi, lorsque les joueurs se connectent soit à l'Hôte soit au server via le menu, le *NetworkManager* se charge de générer un joueur (si besoin) et de charger la scène BossRush.

## 5.5 Site Web

Il y a quelques changements à faire au niveau du texte, les images du jeu a ajouté et le jeu a ajouté en téléchargement, mais la structure du site web est terminée. Nous l'avons publié à l'aide des **Github Pages** sur un compte dédié à notre jeu.

## 5.6 Avancement du projet

### Tableau d'avancement

	Soutenance n°1	Soutenance n°2
Intelligence Artificielle	10%	60%
Multijoueur	10%	75%
Modèles, Textures & Animations	40%	75%
Level design & Maps	25%	45%
Gameplay : Combat	10%	50%
Gameplay : Énigmes	0%	10%
Site web	75%	95%

## 6 État final du projet Akari

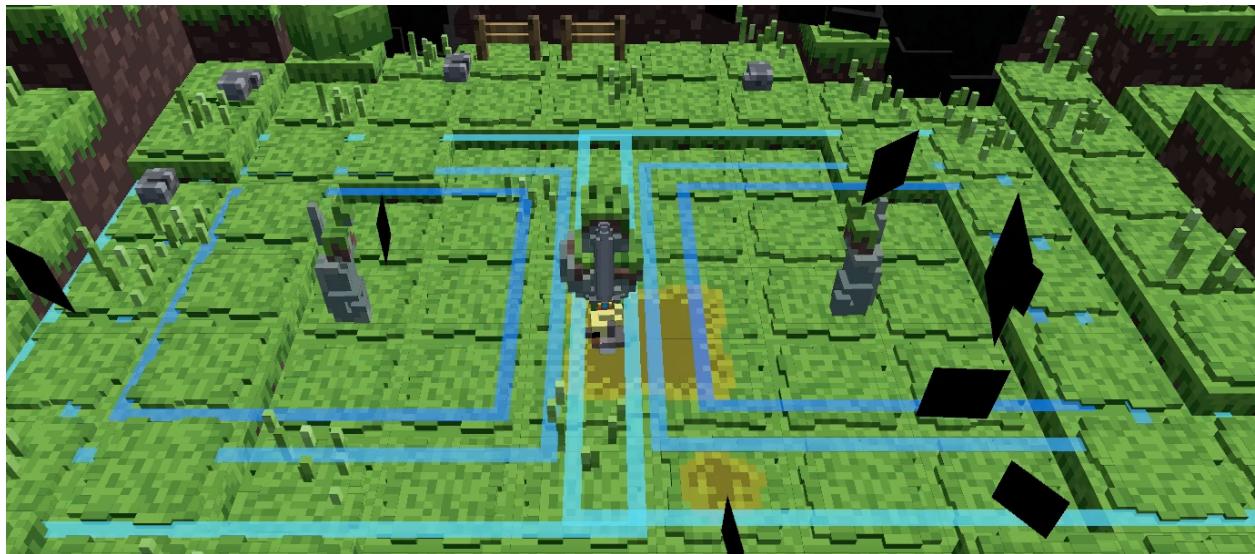
Nous avons maintenant terminé notre projet. Les prochaines parties présentent les nouveautés apportées depuis la deuxième soutenance mais aussi notre vision qui a changé sur certains points.

### 6.1 Entités

Pour les entités, on a maintenant 4 entités finies et fonctionnelles : le **StoneGuardian**, le **SandGuardian**, le **StoryTeller** et les **Panneaux**. Nous avons décidé de supprimer le squelette de notre jeu car il ne rendait pas bien dans les zones que nous avons créés.

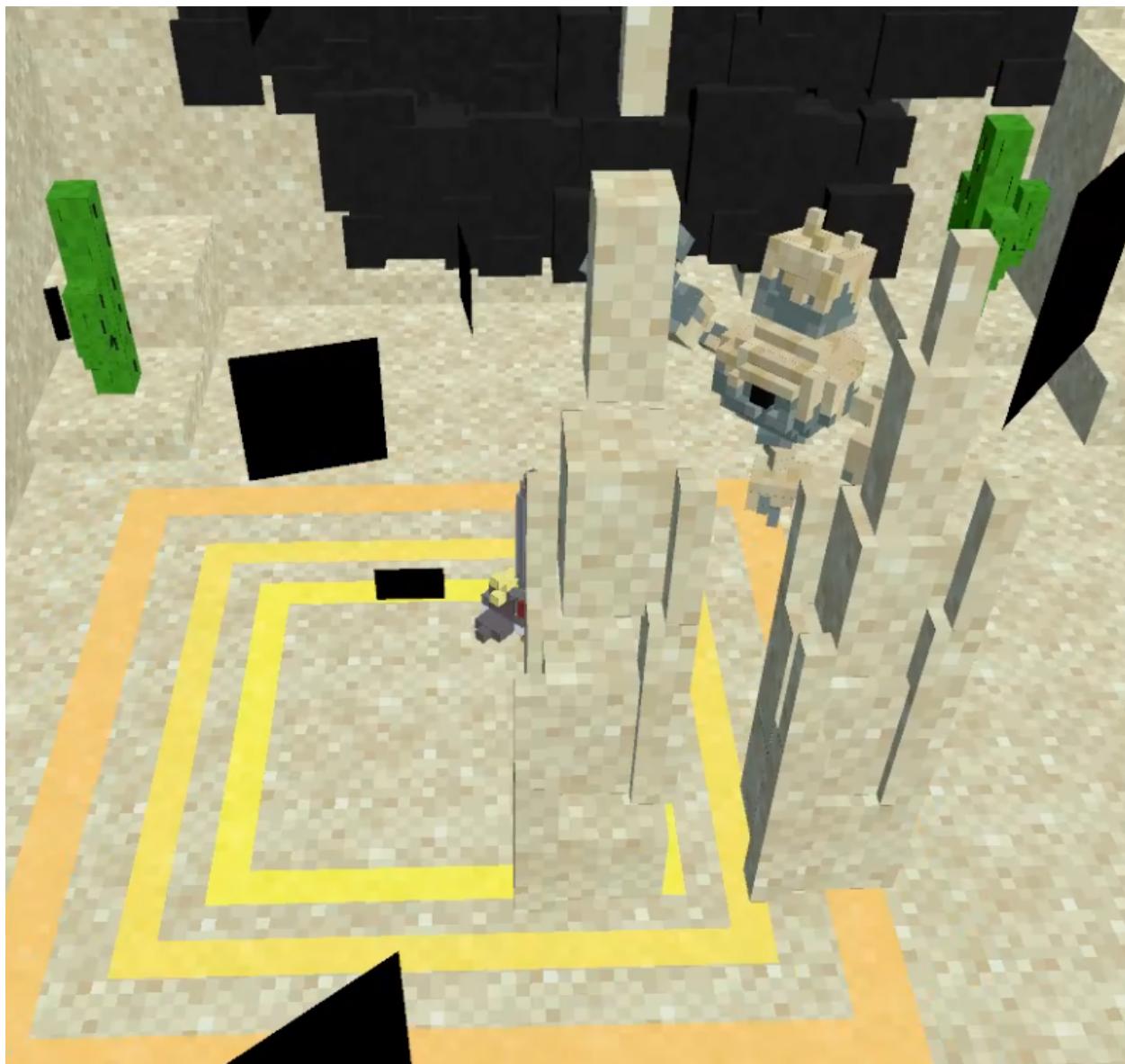
#### 6.1.1 StoneGuardian

Comme raconté précédemment, le StoneGuardian est le monstre à battre dans la zone des plaines pour supprimer la zone de ténèbres. Nous avons retravailler certaines de ces attaques pour offrir une meilleure expérience au joueur. Ce qui était précédemment l'**Attque lourde** est maintenant la **Shockwave**, une attaque de zone, qui rend le StoneGuardian vulnérable.



### 6.1.2 SandGuardian

Le SandGuardian est une version retravaillé du StoneGuardian. Il présente une forme adapté à sa zone, par le sable sur lui. Comme le StoneGuardian, c'est lui le monstre à vaincre. Il présente les mêmes attaques que son semblable, ainsi qu'une nouvelle attaque : **les Piliers de sables**. Le SandGuardian fait apparaître des piliers de sable sous le joueur pour lui infliger des dégâts. Le joueur peut l'anticiper grâce à la **zone rouge** se rétécissent là où le pilier va apparaître.



### 6.1.3 StoryTeller

Le StoryTeller est le premier personnages que l'on croise et qui nous raconte l'histoire du monde ainsi qu'un tutoriel sur les déplacements.



### 6.1.4 Panneaux

Les Panneaux sont des entités avec laquelle le joueur peut interagir avec pour faire apparaître une boîte de dialogue. Ils servent de tutoriels au joueur mais peuvent aussi servir à raconter l'histoire.



## 6.2 Maps

Avant la deuxième soutenance, nous avons décidé de réduire nos 4 zones annoncées à 2 : les **Plaines** et le **Désert** pour plus se consacrer à la partie gameplay de notre jeu et ne pas avoir un jeu immense mais vide.

### 6.2.1 Plaines

La zone de plaines était déjà prête lors de la première soutenance. Nous avons ajouter des éléments d'énigmes ainsi que notre StoneGuardian entièrement fini. Nous avons retravailler les maps pour y intégrer les nouveaux éléments de jeu.

### 6.2.2 Désert

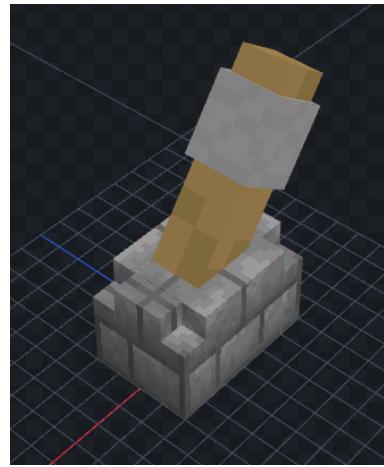
La zone de désert contient une zone de combat contre le SandGuardian. Nous avons utiliser des nouveaux modèles pour le terrain. Il y a aussi des blocs de sables formant un chemin, comme pour les blocs d'herbes. Nous avons aussi des cactus pour une meilleure ambiance.

### 6.3 Énigmes

Notre jeu comporte désormais des éléments d'énigmes que le joueur doit utiliser pour récupérer une récompense.

#### 6.3.1 Levier

Le levier permet au joueur de modifier des éléments de la zone prédéfinis pour accéder à une récompense.



#### 6.3.2 Récompense

Cette récompense sert de *quête secondaire* pour les joueurs voulant tout découvrir.



## 6.4 Gameplay

Nous avons maintenant un jeu jouable dans son ensemble avec un système de combat pouvant résulter sur une mort.

### 6.4.1 Combats

Nos combats se passe en temps réel. Le joueur doit esquiver les attaques des monstres et trouver le bon moment pour attaquer celui-ci. Le monstre possède différentes attaques et peux prendre un certain nombre de dégâts avant de mourir. Tout comme les monstres, le joueur peut attaquer avec son épée et prendre des dégâts jusqu'à en mourir.

### 6.4.2 Mort

Lorsque le joueur prend trop de dégâts, celui-ci meurt. Si cela arrive, le joueur voit apparaître un menu pour lui indiquer qu'il a perdu et pour lui demander si il veut continuer ou arrêter.

## 6.5 Sauvegarde

Notre jeu possède un système de sauvegarde automatique lorsque le joueur quitte le jeu. Si le joueur revient au menu, le jeu va sauvegarder sa partie pour lui permettre de reprendre plus tard. De même si le joueur éteint le jeu brutalement, par un Alt-F4 par exemple. Le jeu sauvegarde la **scène** dans laquelle il se trouve, sa **position** dans la scène et les possibles changements que peut subir la scène comme la **mort d'un Gardien**.

## 6.6 Avancement du projet

### Tableau d'avancement

	Soutenance n°1	Soutenance n°2	Soutenance n°3
Intelligence Artificielle	10%	60%	100%
Multijoueur	10%	75%	100%
Modèles, Textures & Animations	40%	75%	100%
Level design & Maps	25%	45%	100%
Gameplay : Combat	10%	50%	100%
Gameplay : Énigmes	0%	10%	100%
Site web	75%	95%	100%

## 7 Synthèses personnelles

### 7.1 Marc-Aurèle LÉPINE

Tout d'abord, ce projet m'a permis de découvrir un tout autre aspect de la programmation objet, basée sur un framework. On ne s'en rend pas forcément compte au début, mais c'est presque comme si nous devions réapprendre à coder. Il faut se familiariser avec les bonnes pratiques du framework afin de ne pas perdre trop de temps, il faut faire beaucoup de recherches car il y a une infinité de façons de faire telle ou telle chose, et trouver la meilleure qui colle à la base que l'on a déjà n'est pas chose aisée.

Aussi, en ma qualité de chef de projet, j'ai appris à gérer une équipe de travail ainsi que les aléas qui vont de paire avec le travail d'équipe. Je me suis vite rendu compte que le fait de s'auto imposer des deadlines chaque semaine par exemple, pour faire le point sur le travail réalisé, permettait un avancement plus efficace.

Ce fut une bonne expérience, bien que nous n'ayons pas pu intégrer tout ce que nous aurions voulu.

### 7.2 Damien MARRASSÉ

Pour ma part, j'ai trouvé ce projet très intéressant. Il m'a permis de découvrir de nouvelles choses au niveau du code mais aussi sur le plan personnel. Je me suis rendu compte que j'aime beaucoup travailler sur la partie créative d'un projet. L'organisation a été dure à gérer à cause de tout ce qu'on devait faire en même temps.

### 7.3 Alexandre SANTACREU

Ce Projet de S2 au sein d'EPITA m'aura permis d'avoir une expérience sur pleins d'aspects qui n'auraient pas été abordés si celui-ci n'avait pas eu lieu. Pour citer le principal, je dirai : le travail de groupe avec ses aspects positifs et négatifs ou la recherche en autonomie lors de la rencontre de problèmes divers et variés.

C'est un sentiment positif qui en résulte, notamment avec la sensation d'avoir créé quelque chose de personnel au cours de l'année.

### 7.4 Luca ANDRE

Au cours de ce projet S2 à l'EPITA, j'ai été immergé non seulement dans la sphère du développement web, mais également dans la complexité de la gestion de projet à grande échelle.

La partie développement web a été pour moi une occasion unique de mettre en pratique mes compétences acquises en programmation. J'ai découvert l'importance des détails dans le codage, ainsi que la nécessité d'une conception structurée pour assurer un fonctionnement optimal du site.

Ce projet m'a ainsi donné une vision globale de ce qu'est le développement d'un projet informatique. Il ne s'agit pas uniquement de coder, mais aussi de planifier, de s'adapter et de collaborer. Malgré les défis rencontrés, c'est une expérience dont je ressors plus compétent et préparé pour mes futurs projets.

## 8 Conclusion

Nous sommes plutôt fiers de pouvoir proposer notre projet tel qu'il est aujourd'hui et même si nous étions partis avec des étoiles plein les yeux et que nous avons dû retirer de nombreuses idées de notre projet, le résultat final reste néanmoins satisfaisant et l'expérience que nous en tirons est grandissante.

La chose la plus intéressante de ce projet aura sûrement été de pouvoir mettre en commun nos idées et notre travail de manière optimale pour créer quelque chose de personnel en équipe, on en retient donc tous le sentiment d'une expérience enrichissante.