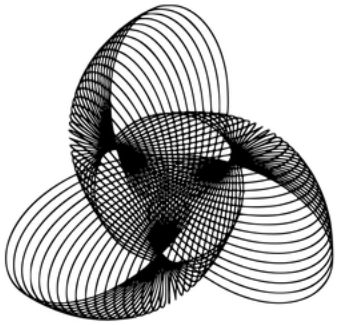


EST. 2023

UNUNUNIUM

COMPANY

Rapport de soutenance
« Les maisons aux âmes perdues »

BERNARDEAU Lucas
JULIA Guillaume
BOCAGE Célia
BRAULT Valentin
VANHAESEBROUCKE Marius

19 mars 2024

Table des matières

1 Introduction

Ce rapport de soutenance a pour but de vous présenter l'un des premiers jeux d'une grande future série imaginée par l'entreprise Unununium. Intitulé "Les maisons aux âmes perdues", le jeu vous plonge dans un univers cartoonesque mélangeant horreur et énigmes tout en vous offrant la possibilité de vous amuser seul ou accompagnée d'un ami.

Bien évidemment, notre objectif principal est que "Les maisons aux âmes perdues" correspond aux attentes du jury, ce qui nous permettra sûrement de développer une version complète de notre jeu. Puisqu'en l'état, nous allons présenter seulement le premier niveau de notre jeu qu'en tant que démonstration.

Notre jeu émane de l'idée de fabriquer de toute pièce un escape game. De plus, au vu de l'attrait certain des membres de l'équipe pour le monde du mystique et de la peur, nous avons décidé que c'est de cette ambiance que s'inspirera notre jeu. C'est pourquoi le premier jeu d'Unununium sera un mélange entre jeu d'horreur et jeu d'énigmes.

"Les maisons aux âmes perdues" se veut être un jeu d'action et d'aventure. Mais aussi un jeu de réflexion. En effet, de par son gameplay attrayant et les nombreuses énigmes du jeu. Vous serez dans l'obligation de vous y amuser en réfléchissant aux énigmes posées par le fantôme. Vous vivrez l'aventure d'un journaliste a la quête de nouvelle histoire mystérieuse a raconté. Enfermé dans la maison, vous devrez alors tenter de vous enfuir tout en échappant au fantôme. Ce dernier ne voulant pas que vous l'abandonniez de ci tôt. Pour parvenir à vous échapper, vous devrez donc résoudre les nombreuses énigmes afin de trouver l'ultime clé qui vous permettra de vous enfuir.

L'Intelligence artificielle de notre jeu sera incarnée par le fantôme. Cette dernière aura pour uniques tâches d'attaquer et de poursuivre le ou les joueur. En effet, vous pourrez jouer avec un ami. Notre jeu sera doté d'un multijoueur en coopération limitée a deux joueurs. Pour des raisons de facilité, nous avons choisi d'effectuer le multijoueur en LAN (Local Area Network, ou réseau local, désigne les appareils connectés, par Wi-Fi ou connexion filaire, dans un domicile ou bureau).

En outre, nous avons choisi que notre jeu serait en 3D afin de vivre au mieux l'expérience de la poursuite du fantôme et de jouer tout en aillant des frissons. Nous utiliserons des musiques et des effets sonores près existant, mais libres de droit. De plus, nous créerons nous-mêmes notre site internet.

2 L'entreprise

2.1 Unununium

Nous sommes une entreprise qui s'est formée le 27 septembre 2023 lors de notre sortie d'EPITA, soit trois mois après avoir été diplômés. Nous nous sommes regroupés autour des passions du jeu vidéo et de l'aventure. Tous passionnés par les énigmes, notre secteur d'activité principale est celui des jeux d'énigmes et d'aventures. Nous sommes une entreprise basée sur le partage et la joie. Toutes nos activités sont créées en ce sens. Nous organisons régulièrement des activités pour renforcer les liens qui nous unissent, comme des "escape game" ou des jeux grandeur nature, par exemple le jeu du "Loup Garous" [1]. C'est ainsi que l'idée de créer "Les maisons aux âmes perdues" est née, mais nous vous en parlerons plus précisément dans l'origine de notre projet.

Unununium peut paraitre être un nom assez drôle, mais il a du sens pour nous.
Lors de la création de nos premiers jeux, nous avons rencontré quelques problèmes dû aux systèmes d'exploitation (d'un point de vue des graphismes).
En nous documentant, nous avons découvert que d'autre créateur avait été confronté au même problème. Néanmoins, certains se sont penchés sur le sujet en créant un système d'exploitation en temps réel graphique à composant.
L'équipe du projet décrivait ainsi ses objectifs : "Nous voulons un système complètement dynamique, où absolument aucun composant individuel n'est requis en mémoire de façon permanente, où la vitesse des applications en cours d'utilisation est de première importance, et où les programmeurs ont un total contrôle de la machine" [2].
Ce projet était temporairement abandonné en mars 2007, pour reprendre en juin 2009. À nouveau abandonné en 2013 [3].
Place maintenant à la présentation de l'équipe en charge du projet.

2.2 Les membres de l'équipe

Lucas BERNARDEAU
- Poste : chef de projet et chargé du web et du Gameplay
- Surnom : Le président
- Devise : "La peur n'arrête pas le danger." [4]

Il est âgé de 18 ans. Il est passionné par l'informatique depuis le lycée. Il a commencé par coder de petits logiciels pour pirater son ancien ordinateur portal. Puis, il a découvert le monde du jeu vidéo. Cet autre univers de l'informatique lui a permis d'exprimer pleinement sa créativité. Il a commencé par créer des jeux comme "Le pendu", "Devine mon nombre entre 0 et 100" ou bien le "Tic-Tac-Toe" en python.
À sa sortie d'EPITA, il décide de créer une entreprise dans le jeu vidéo en recrutant ses anciens camarades de classes avec qui il entretient de bonnes relations. Ses capacités en programmation (C#, python, HTML/CSS, LaTex) lui permettent d'intervenir et de superviser le bon développement du projet.

1. Jeux de société : par Philippe des Pallières et Hervé Marly, ainsi que par les éditions Lui-même.
2. Source : Wikipédia
3. Source : Wikipédia & Dépôt forge (https ://sourceforge.net/projects/uuu/)
4. Eros Bonio : champion de France kickboxing et frère d'Enzo Bonio, double champion du monde de kickboxing

Valentin BRAULT

— Poste : Chargé des menus et de L'Intelligence Artificielle
— Surnom : Nouille
— Devise : Rien n'est impossible.

Il est âgé 18 ans. Il est passionné d'informatique depuis son adolescence. Il découvre l'informatique dans un accompagnement en seconde et depuis il continue de coder des petits projets. En effet, il programme des mini-jeux du type "Snake" ou encore de type "morpion" en Python. C'est un passionné du jeu vidéo. Rejoindre Unununium était pour lui une évidence.
Cette entreprise, spécialisée dans divers types de jeux vidéo, lui correspondait parfaitement.
De plus, cette entreprise était dans son entourage une entreprise réputée tant pour ses conditions de travail que son ambiance. L'entreprise propose également des formations accompagnées et des activités d'intégration. En tant que jeune développeur, cet environnement de travail est un atout.

Marius VANHAESEBROUCKE

— Poste : Chargé du multijoueur et du réseau
— Surnom : Buveur de café
— Devise : Nous sommes notre propre limite.

Depuis son plus jeune âge, il est passionné par les jeux vidéos. Il est curieux, il sait créer un jeu et surtout le rendre excellent et captivant.
Avant d'intégrer à l'EPITA, il faisait seul des petits projets comme des serveurs hébergés sur son ordi pour pouvoir jouer avec des amis. Il peut appliquer dans ce projet ses connaissances en C#, en Godot, en réseau ou encore en Blender. Ce projet est aussi pour lui l'occasion de faire son premier jeu. Il y a longtemps qu'il y songeait.

Célia BOCAGE

— Poste : Chargée du Gameplay et des Graphismes
— Surnom : Cana
— Devise : Force, courage et honneur.

Elle est une programmeuse diplômée de l'EPITA. Elle a grandi à la campagne avec ses deux frères également ingénieurs en informatique. Elle s'est rapidement intéressée à ce domaine et aux compétences qui s'en approchent. Adolescente, elle décide d'investir dans une tablette graphique pour apprendre à manipuler les logiciels de graphisme, autant pour du jeu vidéo ou même de l'animation.
Grâce à ses compétences graphiques. Après ses études, madame Bocage rejoint l'équipe Unununium en 2023. Elle fera en sorte d'aider du mieux possible ses anciens camarades d'EPITA, nouvellement collègues, afin de créer le meilleur jeu d'escape room possible.

Guillaume Julia

— Poste : Chargé du Gamedesign et de la musique
— Surnom : DL-Guigz-16
— Devise : "Hier est derrière, demain est mystère, et aujourd'hui est un cadeau, c'est pour cela qu'on l'appelle le présent." [5]

Il est passionné par l'horreur. Il adore les jeux vidéos. Il porte plus particulièrement son intérêt sur les jeux d'énigmes et d'investigations. Il participe souvent à des escape game. Grâce à son expérience dans les escape game, il s'occupera principalement du gameplay et participera aussi à la modélisation 3D.
À la sortie d'EPITA, il a choisi de rejoindre l'entreprise. Celle-ci lui correspond tant par l'ambiance de travail que par les grandes possibilités de création qu'offrent chaque projet.

5. Film : Kung Fu Panda

3 L'avancement technique

3.1 Le Gameplay

Notre objectif principal était de créer un jeu facile à prendre en main, offrant une expérience fluide et intuitive aux joueurs. Pour ce faire, nous avons axé l'ensemble du développement du gameplay sur la simplicité et l'accessibilité.

En phase de développement, nous avons concentré nos tests sur un environnement Windows et un clavier AZERTY.

Ce choix s'est avéré pertinent pour plusieurs raisons :
— La facilité de configuration et de test : Windows est un système d'exploitation largement répandu et offre une grande compatibilité avec les jeux vidéo.
— La standardisation des touches : Le clavier AZERTY est le format le plus utilisé en France, ce qui facilite la prise en main pour la majorité des joueurs.

Néanmoins, notre ambition ne s'arrête pas là. La version finale du jeu sera accessible sur tous les environnements et types de claviers.

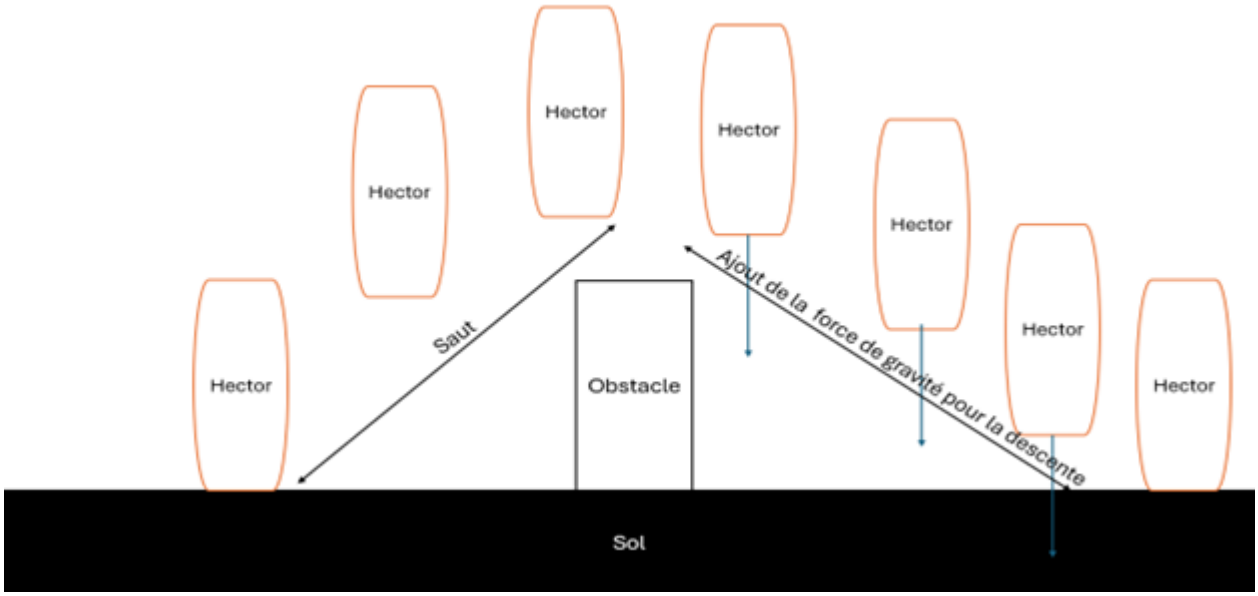
Pour garantir une expérience optimale sur toutes les plateformes et configurations, nous avons mis en place un processus de développement itératif. Ce processus implique :
— L'intégration progressive des fonctionnalités : Ajout et test des fonctionnalités de manière progressive pour identifier et corriger les problèmes potentiels.
— Des tests utilisateurs réguliers : Réalisation de tests avec des joueurs pour recueillir leurs avis et suggestions.
— L'adaptation du gameplay : Ajustement du gameplay en fonction des résultats des tests pour garantir une prise en main facile et intuitive.

La simplicité et l'accessibilité sont des piliers fondamentaux de notre jeu. En privilégiant une approche de développement itérative et en tenant compte des retours des joueurs, nous sommes convaincus de proposer une expérience de jeu agréable et accessible à tous.

Pour notre gameplay, nous avons commencé par programmer notre joueur. Nous lui avons ajouté les déplacements les plus importants tels qu'avancer, reculer, aller à gauche et à droite. Nous avons instancié une nouvelle classe que nous avons nommée « Player ». Nous lui avons ajouté un corps et une tête pour gérer les collisions. Nous développerons ultérieurement la gestion des collisions. Nous nous sommes servis d'un vecteur à trois dimensions pour coder les déplacements puisque nous sommes en 3D. Lors de l'appui des touches pour avancer et reculer, respectivement « Z » et « S », nous faisons varier un second vecteur à deux dimensions sur l'axe des Y. Pour se déplacer de gauche à droite, respectivement avec les touches « Q » et « D », nous faisons varier ce second vecteur sur l'axe des X. Nous mettons alors le vecteur à trois dimensions à jour en lui ajoutant les données. du vecteur à deux dimensions. Enfin, nous nous sommes servis de fonctions déjà présentes dans Godot pour capturer les mouvements de la souris et permettre au joueur de tourner la tête tout en se déplaçant.

Dans la mesure où les actions de déplacements étaient fonctionnelles, et que les chargés de la 3D avaient terminé leur partie, nous avons décidé d'ajouter directement les animations et le modèle 3D au personnage, le but étant de gagner du temps par la suite.

En somme, nous avons ajouté les actions de « saut » et de « position accroupie » directement liées à leurs touches respectives « ESPACE » et « SHIFT ». Nous vous rappelons que nous avons créé précédemment un corps et une tête. C'est à partir de ce moment précis qu'ils nous seront utiles. Pour le saut, nous n'avons qu'augmenté l'axe des Y en lui ajoutant la force de gravité (arrondi a 9.81 pour notre programme) pour qu'il puisse retomber sur le sol à la fin de son saut.



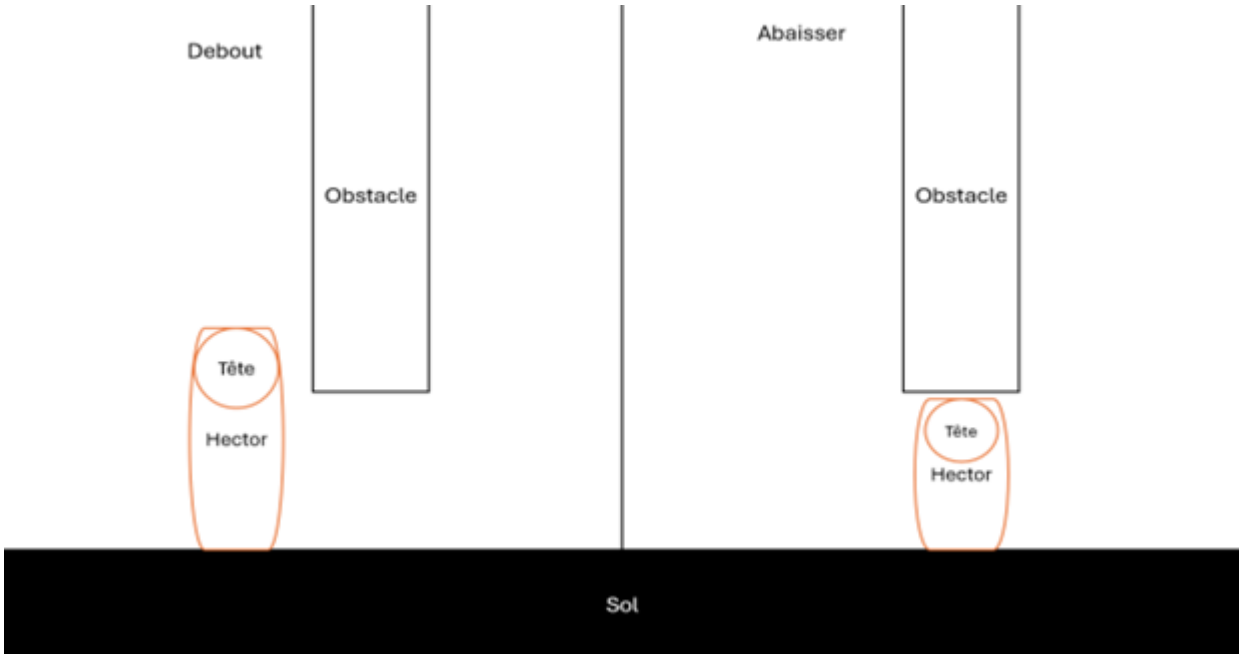
Le code associé :

```
if (Input.IsActionPressed("space") && IsOnFloor())
    velocity.Y = jumpForce;
}
//on normalise deplacement
direction = direction.Normalized();
var forward = GlobalTransform.Basis.Z;
var right = GlobalTransform.Basis.X;
//avant/arriere
velocity.Z = (forward * direction.Y + right * direction.X).Z * moveSpeed;
//gauche/droite
velocity.X = (forward * direction.Y + right * direction.X).X * moveSpeed;
//haut/bas/saut et ajout de la gravite
velocity.Y -= gravity * (float)delta;
//on applique mouv
Velocity = velocity;
MoveAndSlide();
```

La position accroupie du personnage est une mécanique importante dans notre jeu.
Pour l'implémenter, nous avons combiné deux techniques :

- La modification de la position de la tête :
  - Le déplacement de la tête sur l'axe Y (vertical) pour simuler l'inclinaison du corps.
  - L'ajustement de la position du centre de gravité du personnage pour maintenir son équilibre.
- La réduction de la taille du corps :
  - La diminution de la hauteur du corps pour simuler l'accroupissement.
  - L'adaptation des collisions pour correspondre à la nouvelle taille du personnage.

Ces techniques combinées offrent plusieurs avantages :
— Le réalisme accru : L'inclinaison de la tête et la réduction de la taille du personnage contribuent à un rendu visuel plus réaliste de l'abaissement.
— La fluidité du mouvement : La modification du centre de gravité permet de maintenir un mouvement fluide et naturel du personnage.
— Une meilleure gestion des collisions : L'adaptation des collisions à la nouvelle taille du personnage garantit une interaction précise avec l'environnement.



La position accroupie du personnage est une fonctionnalité importante qui enrichit le gameplay et l'immersion du joueur. La combinaison de la modification de la position de la tête et de la réduction de la taille du corps s'est avérée efficace pour obtenir un résultat visuel et technique satisfaisant.

Notre jeu d'horreur se déroule dans l'obscurité. La lampe torche devient un élément crucial pour la survie du joueur.

Pour l'implémenter, nous avons :
— Assigné la touche "E" à l'activation/désactivation de la lampe.
— Utilisé deux booléens dans le programme pour gérer la visibilité de la lampe.

Pour enrichir les interactions du joueur, nous avons ajouté :
— Des points de vie pour gérer les interactions avec l'intelligence artificielle et les ennemis.
— Une liste servant de référence au futur sac à dos du personnage, lui permettant de stocker des objets et d'accéder à de nouvelles actions.

Grâce à ces implémentations, notre personnage dispose désormais des fonctionnalités essentielles pour évoluer dans l'univers du jeu :
— Se déplacer dans l'obscurité grâce à la lampe torche.
— Survivre aux ennemis en gérant ses points de vie.
— Interagir avec l'environnement
— Collecter des objets grâce au sac à dos.

Après la mise en place des bases du jeu, nous avons introduit les premiers objets, en commençant par une classe "médicaments". Cette classe permet au joueur de restaurer ses points de vie perdus face à l'IA (représentés par un fantôme).

Une fonction permet au joueur d'utiliser la classe « médicaments » à condition qu'il en possède déjà dans son inventaire. La fonction lui augmente ses points de vie.

Cette fonction s'articule autour de plusieurs étapes :

1. La détection de l'utilisation d'un médicament : Le joueur doit d'abord utiliser un objet de la classe "médicaments" via une interaction définie (par exemple, une touche du clavier).
2. La vérification de la possession d'objets : Le jeu vérifie si le joueur possède vraiment des médicaments dans son inventaire.
3. L'augmentation des points de vie : Si la vérification est positive, la fonction augmente les points de vie du joueur en fonction d'une valeur prédéfinie.
4. La mise à jour de l'interface : L'interface du jeu est mise à jour pour afficher le nouveau nombre de points de vie du joueur.

Le code associé :

```
public bool Health()
{
  if (sacamedicament.Count < 3)
  {
    sacamedicament.Add(new medicament_soins());
    return true;
  }

  return false;
}
```

L'ajout de la classe "médicaments" et de la fonction de regain de santé a eu un impact positif sur le gameplay comme :

— L'ajout d'une dimension stratégique : Les joueurs doivent désormais gérer leur inventaire et choisir le bon moment pour utiliser les médicaments.
— L'augmentation de la durée de vie du joueur : La possibilité de restaurer des points de vie permet aux joueurs de survivre plus longtemps et de progresser davantage dans le jeu.
— La création d'une expérience plus immersive : La gestion de la santé et l'utilisation d'objets contribuent à une expérience de jeu plus réaliste et engageante.

L'intégration des objets et des fonctionnalités de santé s'est avérée une étape importante dans le développement du jeu. Cette évolution a enrichi le gameplay, apporté une dimension stratégique et contribué à une expérience de jeu plus immersive et stimulante.

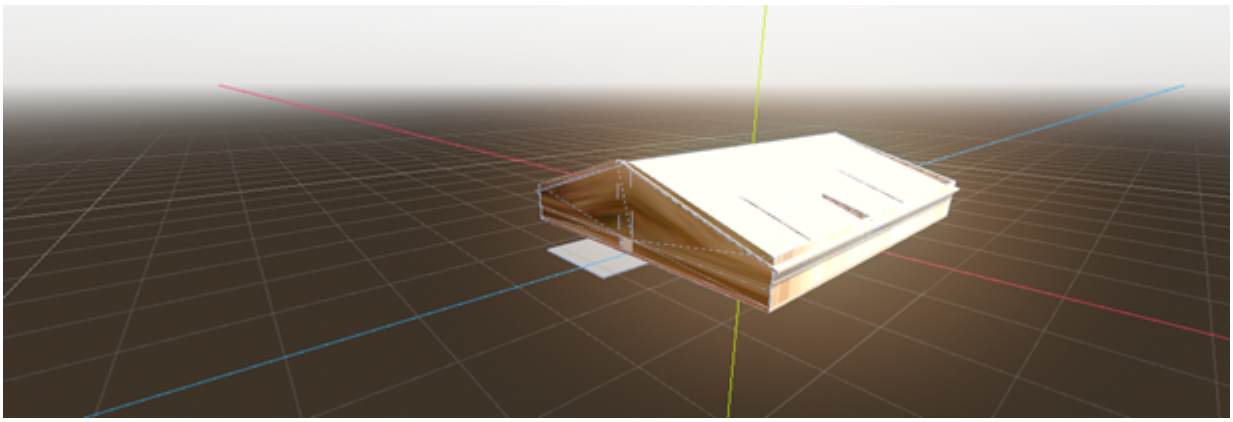
La maison est un élément central de notre jeu. C'est le lieu où se déroule l'intégralité de l'action. Son design et sa fonctionnalité étaient donc des aspects cruciaux à prendre en compte.

Pour créer une maison à la fois esthétique et fonctionnelle, nous avons collaboré étroitement avec les membres de l'équipe en charge du design. Ils ont apporté leur expertise et leur créativité pour donner vie à un environnement immersif et cohérent avec l'univers du jeu.

L'implémentation de la maison n'a pas été sans défis. L'un des points les plus problématiques concernait les escaliers. En effet, leur conception et leur intégration dans l'environnement se sont avérées plus complexes que prévu. Ces difficultés et les solutions que nous avons mises en place sont décrites dans notre rapport.

Malgré les défis rencontrés, nous sommes convaincus que la maison est un élément essentiel qui contribuera à l'expérience de jeu immersive que nous souhaitons offrir aux joueurs.

Nous sommes fiers du travail accompli. La collaboration entre les équipes de développement et de design a permis de créer un environnement immersif et fonctionnel qui contribuera à l'expérience de jeu des joueurs.



3.2 Le Game design

Le game design est une combinaison entre interactions, énigmes et maniement du joueur. On a donc rajouté une touche "interagir" qui va nous être utile pour les différentes interactions, par exemple, nous avons la clé qui va nous permettre tout au long du niveau d'être une récompense pour les énigmes et de donner accès aux différentes salles de la maison.

Extrait de code de la clé :

```
public partial class doorame : Area3D
{
    private Area3D CLE;
    private bool Dedans = false;

    public override void _Ready()
    {
        CLE = GetNode<Area3D>("/root/Node3D/Kee");
    }

     public override void _Process(double delta)
    {
        if (Input.IsActionPressed("interact") && Dedans == true)
         {
           CLE.Hide();
        }
    }
    private void OnCollision(Node3D body)
    {
       Dedans = true;
    }
    private void out_collision(Node3D body)
    {
     Dedans = false;
    }
}
```

Au début, il y a donc les déclarations nécessaires, on crée une référence à l'objet de type Area3D qui représente la clé. De plus, la variable "Dedans" indique lorsque le joueur est ou non dans la zone de la clé pour la récupérer, la méthode _Ready est appelée lorsque l'instance doorame est prête à être utilisée.

En somme, on récupère la référence à partir de la scène avec GetNode et le chemin de la scène. La méthode _Process est appelée à chaque frame du jeu, ce qui permet de vérifier si le joueur appuie sur la touche d'interaction quand il est à l'intérieur de l'Area3D de clé. Si les conditions sont respectées, l'objet disparaît.

La méthode OnCollision est dérivée de la fonction de base d'Area3D body_entered, elle met donc à jour la variable "Dedans" lorsque le joueur est à l'intérieur de la zone de clé. Et à l'inverse, out_collision est une méthode se basant aussi sur une fonction de base d'Area3D qui est body_exited qui met la variable "Dedans" égale à false lorsque le joueur sort de la zone de la clé. La clé va donc être complétée avec la porte.

En outre, pour mettre en œuvre l'utilité de la clé, il faut l'utiliser dans une énigme. Pour cela, nous avons rajouté des éléments tels qu'un panneau alliant un MeshInstance pour sa forme avec un Label pour son texte et une texture. Tres, le panneau est utiles pour mettre en contexte le début d'une énigme sans être perdu et de pouvoir visualiser certaines étapes.

Nous avons rajouté une lanterne qui va servir à diriger le joueur en attirant son œil, elle récupère les propriétés de la lampe pour la lumière qu'elle émet tout autour d'elle.

3.3 Les Graphismes

L'équipe d'Unununium est fière de présenter les progrès réalisés sur le plan des graphismes pour son jeu vidéo. Notre objectif initial était de créer une carte unique, à l'esthétique cartoonesque et horrifique, parfaitement adaptée à l'univers du jeu.

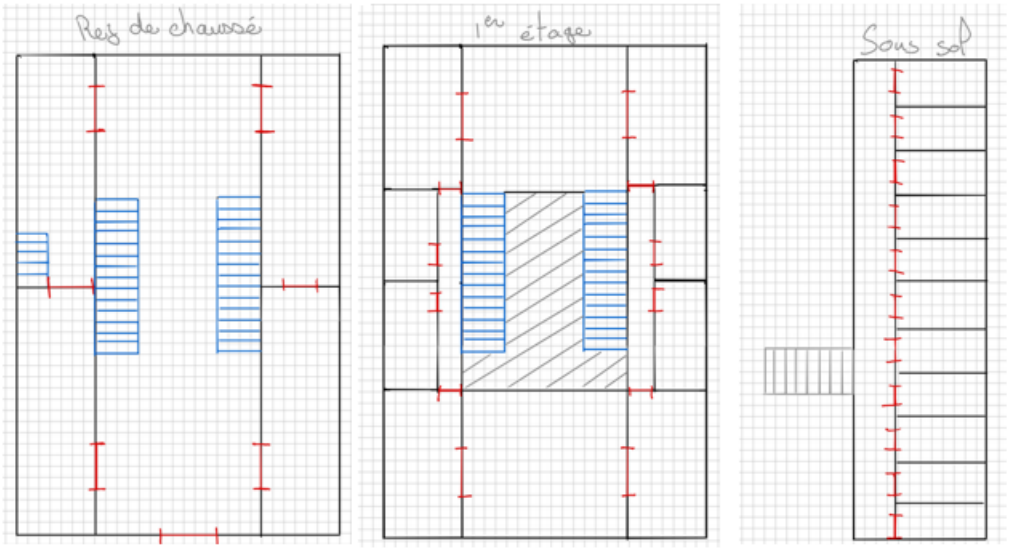
Concernant la map, nous avons opté pour un manoir, environnement riche en potentialités narratives et offrant une grande variété d'espaces à explorer. Après avoir défini le nombre d'étages et le nombre approximatif de pièces, nous avons pu modéliser l'ensemble de manière originale.

Le style visuel du jeu se veut à la fois cartoonesque et horrifique. Cette combinaison permet de créer une ambiance unique, à la fois amusante et effrayante. Les personnages et les décors sont ainsi conçus pour être à la fois drôles et grotesques, créant un sentiment de malaise et d'inquiétude chez le joueur.

L'équipe d'Unununium est consciente des défis que représente la création de graphismes de qualité pour un jeu vidéo. Nous sommes néanmoins déterminés à offrir aux joueurs une expérience visuelle unique et immersive.

Des discussions approfondies ont permis de définir les grandes lignes architecturales du manoir. Un schéma 2D a ensuite été élaboré, illustrant la disposition des pièces, l'organisation des espaces et l'esthétique générale de la bâtisse. Ce schéma a été soumis à l'équipe pour validation et a fait l'objet d'un consensus.

Le schéma suivant a été utilisé comme base pour la construction de la carte :



Fort de l'approbation du plan 2D, la modélisation 3D du manoir a débuté. Cette étape a permis de visualiser le projet en volume et d'affiner certains aspects du design. Des modifications ont été apportées, telles que le repositionnement de portes ou l'ajout de pièces complémentaires, afin d'optimiser l'agencement et la fonctionnalité du manoir.

Le processus de création du plan du manoir s'est déroulé en deux phases distinctes, chacune contribuant à la réalisation d'un projet abouti et répondant aux attentes de l'équipe. La phase de conception 2D a permis de poser les bases du projet et de définir les orientations architecturales majeures. La modélisation 3D a ensuite donné vie au manoir en lui conférant une dimension concrète et en permettant d'en appréhender les volumes et les détails.

Le développement de personnages 3D pour notre projet s'est effectué dans un contexte de contraintes temporelles et de compétences limitées en modélisation sur Blender. Pour garantir la faisabilité du projet et respecter le calendrier défini, nous avons opté pour l'utilisation de modèles 3D libres de droits.

3.4 Les menus

L'implémentation des menus est un élément crucial pour tout jeu vidéo. Ils constituent l'interface principale entre le joueur et l'univers du jeu, offrant une navigation intuitive et un accès facile aux différentes fonctionnalités. Dans le cadre de notre projet, l'équipe s'est fixée comme objectif d'atteindre 60% d'avancement dans la réalisation des menus pour la soutenance. Heureusement, nous sommes fiers d'annoncer que cet objectif a été largement dépassé, avec seulement 20% du travail restant à programmer.

Le menu principal est le point de départ de toute aventure. Il présente trois boutons distincts, chacun menant à une expérience différente :

- Solo : Démarrer une nouvelle partie en solo, où le joueur affrontera les défis du jeu seul.
  - Le code associé :

```
private void _on_start_in_solo_pressed()
{
    player playerInfo = new player() {Name = "SOLO",Id = 1};
    Multi_game_manager.Players.Add(playerInfo);
    var scene =
     ResourceLoader.Load<PackedScene>("res://Gameplay_les_differentes_sce
         nes/scene.tscn").Instantiate<Node3D>();
    GetTree().Root.AddChild(scene);
    this.QueueFree();
}
```

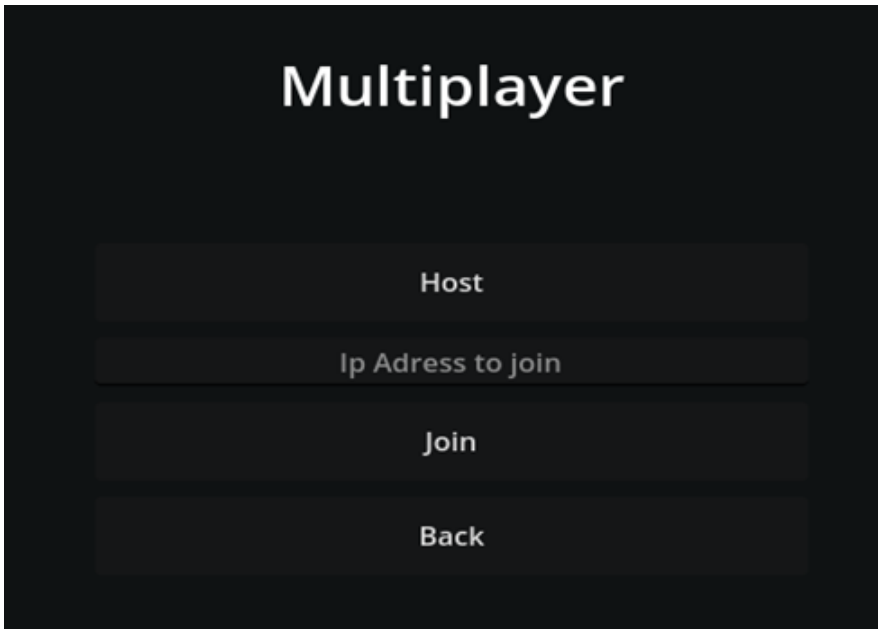
- Multijoueur : Ouvrir le menu dédié au multijoueur pour une partie à deux joueurs.
- Paramétrage des Contrôles : Ouvrir les paramètres de contrôle, mais n'est pas encore en fonction
- Quitter le jeu : Fermer le jeu.
  - Le code associé :

```
private void _on_quit_game_pressed()
{
    GetTree().Quit();
}
```

Chaque bouton a été implémenté avec soin en C#, garantissant une interaction fluide et intuitive. Le code derrière chaque bouton est clair et précis, permettant une maintenance aisée et une extension future.

Le menu multijoueur est l'endroit où les joueurs se connectent et se lancent dans des parties palpitantes à deux. Il propose quatre boutons et un champ de texte pour une configuration de l'adresse IP à rejoindre :

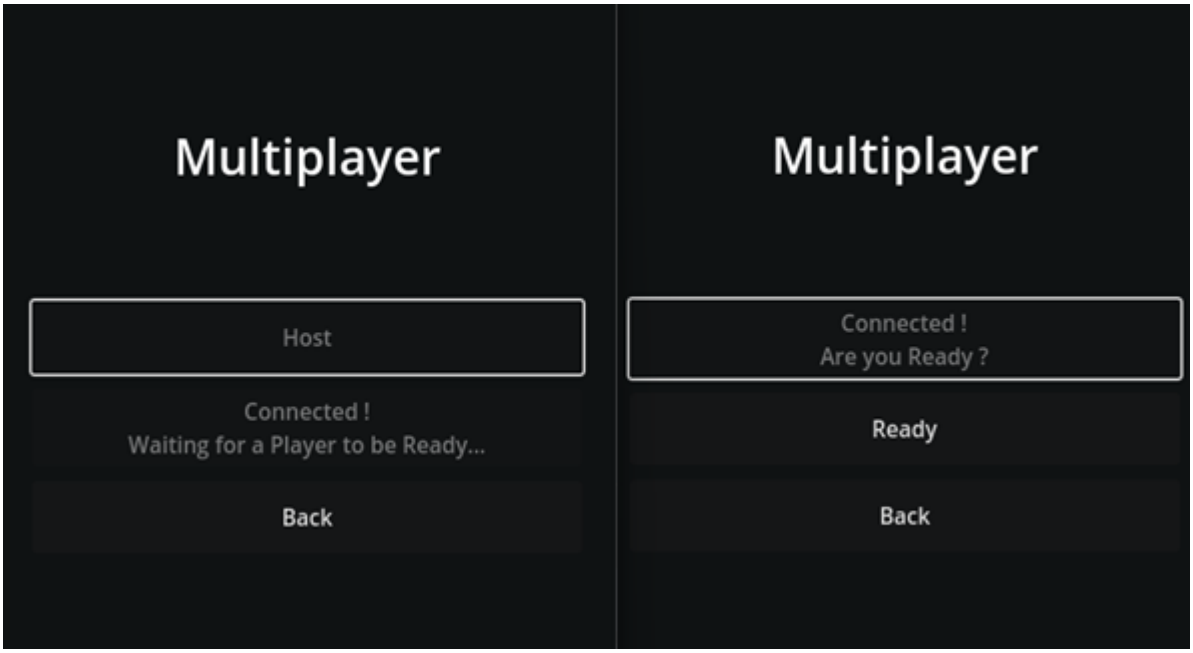
— Héberger : Héberger une nouvelle partie, un serveur est créé et les joueurs en local et connaissant l'adresse IP peuvent rejoindre.
— Adresse IP à rejoindre : Saisir l'adresse IP de la partie à rejoindre pour se connecter.
— Rejoindre : Se connecter à une partie hébergée par un autre joueur, il faut avoir changé l'adresse IP, car l'adresse par défaut est celle locale.
— Retour : Revenir au menu principal et pour arrêter toutes connexions comme le serveur s'il a été créé, si les deux joueurs sont connectés, les renvoient tous les deux au menu principal.



Une fois les deux joueurs connectés :

— Le Client doit d'abord appuyer sur Prêt pour indiquer à l'Host qu'il est prêt à commencer la partie.
— Une fois le Bouton Prêt appuyé, l'Host peut appuyer sur Démarrer pour lancer la partie pour les deux joueurs.

Le menu multijoueur gère également les erreurs de connexion et de communication de manière conviviale et transparente, offrant une expérience fluide et sans frustration.



Accessible en appuyant sur la touche "Echap" pendant le jeu, le menu pause permet de mettre la partie en pause et de prendre un moment de réflexion. Il comporte deux boutons essentiels :

— Continuer : Reprendre la partie là où elle s'est arrêtée.
— Paramétrage des Contrôles : Ouvrir les paramètres de contrôle, mais n'est pas encore en fonction
— Quitter le jeu : Fermer le jeu

Le menu pause est particulièrement utile dans les situations multijoueurs, où il permet aux deux joueurs de mettre la partie en pause simultanément.

Ce menu s'affiche lorsque le joueur perd une partie. Il propose deux options :

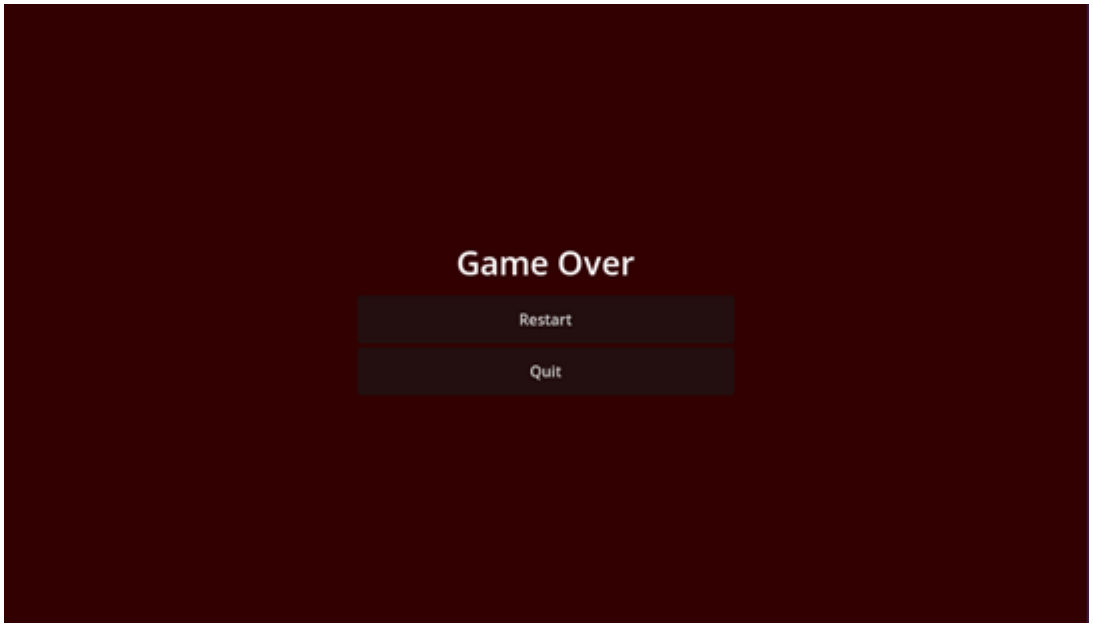
— Recommencer : Démarrer une nouvelle partie immédiatement. En multijoueur, la partie est relancée pour les deux joueurs en même temps.

— Quitter le jeu : Fermer le jeu.

— Le code associé :

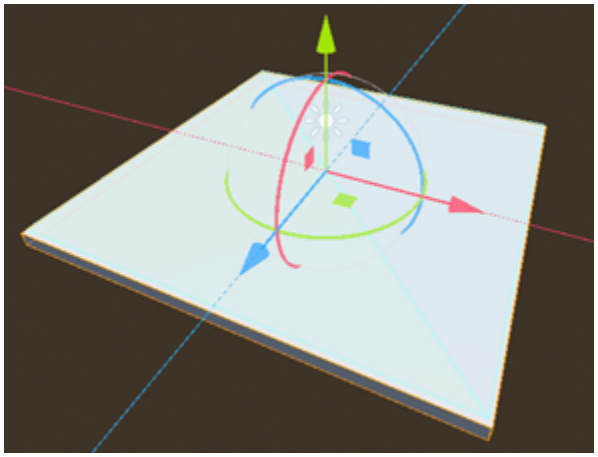
```
private void _on_restart_pressed()
{
    Rpc("RPC_on_restart_pressed");
    RPC_on_restart_pressed();
    QueueFree();
}

[Rpc(MultiplayerApi.RpcMode.AnyPeer)]
private void RPC_on_restart_pressed()
{
    var scene = ResourceLoader.Load<PackedScene>("res:Gameplay_les_dif
      ferentes_scenes/scene.tscn").Instantiate<Node3D>();
    GetTree().Root.AddChild(scene);
    QueueFree();
}
```

Le menu "Game Over" offre une conclusion claire à la partie et incite le joueur à réessayer, s'il le souhaite.

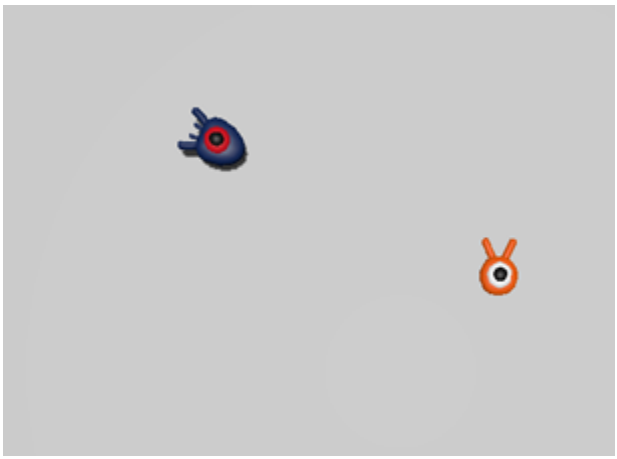


3.5 L'Intelligence Artificielle

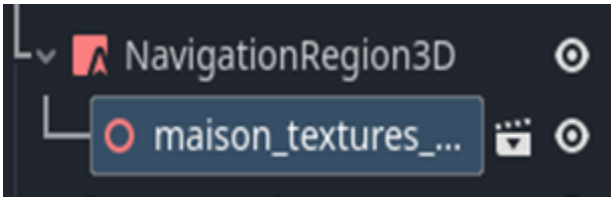
Premièrement, afin de me familiariser avec le moteur de jeu Godot, nous avons suivi le tutoriel « Your first 3D game » de la documentation officielle de ce dernier. Cela a été très important pour la création de l'intelligence artificielle, car en plus de nous apprendre les notions élémentaires de Godot comme les nœuds et les scènes, cela nous a également permis d'avoir un espace de tests pour implémenter les nouveautés de l'IA à petite échelle. Cela fut d'une grande utilité étant donné que le projet devint jour après jour de plus en plus lourd et cela nous a en outre permis d'isoler plus facilement les problèmes rencontrés.



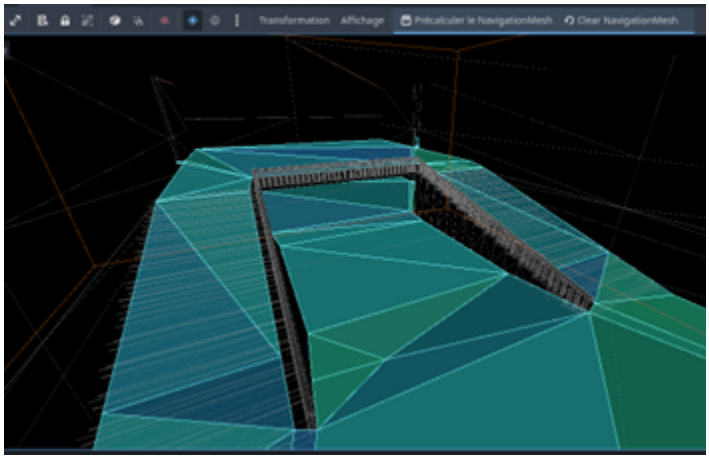
Capture d'écran une fois lancé, mettant en évidence les différents éléments et fonctionnalités de l'espace de travail.



En ce qui concerne les avancements techniques de l'intelligence artificielle, nous avons tout d'abord implémenté la composante principale de notre IA qui est le pathfinding : cela consiste tout simplement au fait que le fantôme suive le joueur.
Pour ce faire, nous avons dû tout d'abord instancier dans la scène principale une NavigationRegion3D en nœud père de la maison.

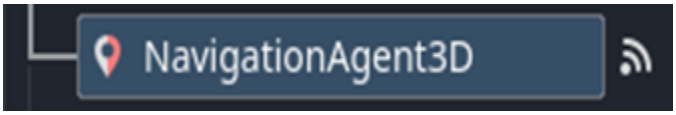


Ce dernier va permettre d'instancier dans la maison un NavigationMesh, qui va agir comme un calque du sol de la maison et va permettre d'indiquer à l'IA où elle peut se déplacer.



Cela est crucial dans notre cas, car nous voulons empêcher notre IA de grimper comme bon lui semble tout type de surface pour atteindre le joueur plus rapidement, ce qui nous le pensons, ajouterait trop de difficulté au jeu.

Deuxièmement, maintenant que l'IA sait dans quelle zone se déplacer, il faut désormais qu'elle sache comment se déplacer vers le joueur. Pour ce faire, nous avons rajouté dans la scène de l'IA, un nœud nommé NavigationAgent3D.



Ce nœud est tout simplement une sorte de GPS pour l'IA. Il va permettre de récupérer la position du prochain point pour aller en direction du joueur, grâce à la fonction GetNextPathPosition(), et donc nous permettre de diriger l'IA vers ce dernier. Pour ce faire, nous avons calculé un nouveau vecteur :

```
Velocity = (vectToPlayer - vectAct).Normalized() * _speed;
```

Nous avons tout d'abord fait la différence entre le vecteur du prochain point et le vecteur de l'IA. Cela permet d'avoir un vecteur qui a comme origine l'IA, comme direction la droite (IA, prochain point) et comme sens l'IA vers le prochain point.
Ensuite, nous avons appelé la fonction Normalized() sur ce nouveau vecteur et nous l'avons multiplié par la vitesse du fantôme. La fonction Normalized() va renvoyer notre vecteur initial, mais avec une norme égale à 1. Cela est essentiel dans ce calcul, car sans cette fonction, la vitesse du fantôme serait proportionnelle à l'écart entre lui-même et le joueur, et donc incontrôlable, ce qui est contraire à ce que nous désirons. En effet, nous souhaitons que le fantôme ait une vitesse constante et également que sa vitesse soit augmentée au fur et à mesure que le joueur résout des énigmes, ce qui requière de pouvoir la contrôler.

Troisièmement, suite à l'implémentation du multijoueur, il fallait désormais que l'IA puisse suivre les deux joueurs, et il fallait donc que l'IA choisisse une cible à suivre. Nous avons fait le choix que l'IA suive le joueur le plus proche, ce qui nous semblait le plus logique. Pour y parvenir, nous avons dû rajouter dans le script de l'IA une fonction ClosestPlayer(). Celle-ci va calculer pour chaque joueur le prochain point vers lequel l'IA doit se diriger. Elle va ensuite comparer les normes de ces deux vecteurs et renvoyer le joueur dont la norme du vecteur calculé précédemment est la plus petite.

```
private CharacterBody3D ClosestPlayer()
{
    if (_player2 == null)
    {
        return _player;
    }
    var vectAct = Position;
    _navAgent.TargetPosition = _player.Position;
    var vectToPlayer1 = _navAgent.GetNextPathPosition();
    var AItoPlayer1 = vectToPlayer1 - vectAct;

    _navAgent.TargetPosition = _player2.Position;
    var vectToPlayer2 = _navAgent.GetNextPathPosition();
    var AItoPlayer2 = vectToPlayer2 - vectAct
    if (AItoPlayer1.Length() < AItoPlayer2.Length())
    {
         return _player;
    }
    return _player2;
}
```

Deuxièmement, nous avons implémenté les interactions entre l'IA et le joueur. Cette section comporte trois nouveaux ajouts :

1. l'IA met à jour le champ de vision du joueur quand ils se touchent.
2. l'IA et le joueur se font téléporter à leurs points d'apparition respectifs quand ils se touchent.
3. le joueur déclenche l'écran « Game Over » lorsque son champ de vision tombe à 0.

Nous allons désormais expliquer ci-dessous comment nous avons implémenté ces différents ajouts : Tout d'abord, pour détecter que l'IA a réussi à atteindre le joueur, mécanique utilisée dans le 1er et le 2e ajout, nous avons utilisé un signal lié du nœud NavigationAgent3D de l'IA nommé target_reached que nous avons lié dans le script du fantôme à la fonction _TargetReached(). Ce signal se déclenche lorsque l'IA aura atteint certaines distances avec sa cible, dans notre cas 1,5 m.

Pour ce premier ajout, il fallait que nous modifions un attribut du joueur. Nous avons donc utilisé un signal envoyé depuis l'IA, dans la fonction GotCaught(), au joueur afin que ce dernier modifie lui-même son attribut dans la fonction _onEnemyCaught().

Script de l'IA :

```
private void GotCaught()
{
    EmitSignal(SignalName.Caught);
}
```

Script du joueur :

```
private void _onEnemyCaught()
{
    Champsdevision-= 1;
    //(reste du code)
}
```

Pour le 2e ajout, nous avons simplement réutilisé les fonctions précédentes pour téléporter les deux personnages. Nous avons téléporté l'IA dans _TargetReached() et le joueur dans _onEnemyCaught() en changement leurs positions aux coordonnées de leurs points de réapparition respectifs.

Script de l'IA :

```
private void _TargetReached()
    {
            Position = _spawnPoint;     //On TP le fantome a son SpawnPoint
            GotCaught();
            Rpc("Synch_Pos", _spawnPoint);
    }
```

Script du joueur :

```
private void _onEnemyCaught()
    {

            Champsdevision-= 1;
            Position = _respawnPoint;
            //(reste du code)
    }
```

Pour le dernier ajout, nous avons ajouté dans la fonction _onEnemyCaught() du joueur un test pour savoir si son champ de vision est descendu à 0. Si c'est le cas, le joueur va envoyer un signal au script de la scène principale pour que celle-ci change la scène.

Script du joueur :

```
private void _onEnemyCaught()
{
    Champsdevision-= 1;
    Position = _respawnPoint;
    if (Champsdevision == 0)
    {
        EmitSignal(SignalName.PlayerDead);
    }
}
```

Enfin, le dernier point a été l'instanciation du fantôme. Pour ce faire, il nous a fallu ajouter au script de la scène principale la ligne suivante :

```
[Export] private PackedScene _phantomScene;
```

Celle-ci permet de récupérer la scène du fantôme que nous instancions dans la fonction _Ready(). Ensuite, nous attribuons au fantôme son nom, pour pouvoir récupérer son nœud dans d'autres scripts, ainsi que sa position de départ qui est son point de réapparition.

3.6 Le Multijoueur & Réseau

Le multijoueur est un excellent moyen de partager l'expérience de jeu et de s'amuser entre amis. C'est pourquoi nous avons mis l'accent sur la création d'un système multijoueur simple à mettre en place pour les joueurs.

Pour la connexion entre les joueurs, nous avons opté pour le réseau local (LAN). Cette solution est simple à configurer et ne nécessite aucun serveur externe. Et pour la synchronisation entre les joueurs, on a utiliser les outils Multijoueur intégré à Godot qui sont très complet et performant.

Une fois que les joueurs sont sûrs que la connexion peut se faire entre leurs ordinateurs grâce à un réseau local ou un réseau local virtuel.

Le joueur qui va héberger la partie devra donner son adresse ipv4 à l'autre joueur pour le rejoindre.

Étant donné que nous utilisons les outils multijoueurs intégrés de Godot, il est crucial de synchroniser les identifiants uniques de chaque joueur. La création de l'identifiant s'effectue lors de l'action "Host" ou "Join".

Le système de synchronisation d' identifiants est conçu pour gérer un nombre illimité de joueurs. Si le nombre de joueurs devait changer à l'avenir, une modification simple de quelques lignes de code suffirait.

Le processus de synchronisation des identifiants a été réalisé en utilisant la méthode suivante :

```
private void ConnectedToServer()
{
    GD.Print("Connected To Server");
    GetNode<Button>("VBoxContainer/Join").Disabled = true;//Pour le Menu
    GetNode<Button>("VBoxContainer/Join").Text =
            "Connected !"+"\nAre you Ready ?";
    GetNode<LineEdit>("VBoxContainer/Ip_Address").Visible = false;

    _name = Multiplayer.GetUniqueId().ToString();
            //pour la synchronisation des id
    SendPlayerInformation(_name, Multiplayer.GetUniqueId());
    Rpc("SendPlayerInformation", _name, Multiplayer.GetUniqueId());
}

[Rpc(MultiplayerApi.RpcMode.AnyPeer)]
private void SendPlayerInformation(string name, int id)
{
    player playerInfo = new player {Name = name,Id = id};
    if (!Multi_game_manager.Players.Contains(playerInfo))
    {
        Multi_game_manager.Players.Add(playerInfo);
    }
    if (Multiplayer.IsServer())
    {
        Rpc("SendPlayerInformation",_name,1);
    }
}
```

Le script utilise la fonction Rpc, cette fonction permet d'exécuter une fonction sur tous les clients connectés. Cela permet aussi d'envoyer des informations entre les joueurs, comme dans l'exemple précèdent.

La scène principale utilise la liste des joueurs et leurs identifiants pour créer des instances de chaque joueur. Les nœuds des joueurs sont ensuite renommés en fonction de leur identifiant, ce qui facilite la synchronisation des positions et des actions entre les clients.

L'outil MultiplayerSynchronizer est utilisé pour synchroniser les positions et les actions des joueurs en temps réel. Cela permet de garantir que tous les clients voient la même chose et que le jeu se déroule de manière fluide.

En outre, pour la synchronisation du fantôme, l'hôte transmet régulièrement la position du fantôme au client. Dans certains cas spécifiques, tels que l'atteinte de la cible par le fantôme, le client peut également envoyer une mise à jour de la position.

Pour illustrer ce propos, voici un exemple de code :

```
private void _TargetReached()
{
    GD.Print("AItouched");
    Position = _spawnPoint; //On teleporte le fantome a son SpawnPoint
    GotCaught();
    Rpc("Synch_Pos", _spawnPoint);
}

[Rpc(MultiplayerApi.RpcMode.AnyPeer)]
private void Synch_Pos(Vector3 pos) //synchronise la position du fantome
{
    Position = pos;
}
```

Menus Pause et Game Over :

Enfin, pour les menus, comme nous avons décidé de garder la pause du jeu en Multijoueurs, il nous a suffi de synchroniser la pause du jeu et l'ouverture et la fermeture du menu pause.

De plus, pour le menu Game Over, comme la mort d'un joueur provoque une fin de partie de l'autre joueur, il nous suffit de synchroniser l'ouverture du menu.

En outre, pour le relancement de la partie, il se fait aussi en même temps pour éviter toute erreur de synchronisation qui pourrait être causée par une instanciation différée des nœuds.

3.7 Le Son

L'un des objectifs principaux de la phase actuelle du développement d'Unununium était l'intégration d'une ambiance musicale. L'ambition était d'atteindre un niveau de complétude de 10% pour cette première soutenance, et cet objectif a été atteint avec succès.

La première étape a consisté à sélectionner une musique d'ambiance libre de droit pour le menu d'accueil. Pour ce faire, une recherche approfondie a été effectuée sur la plateforme OpenGameArt.org. Après une sélection rigoureuse, un morceau répondant parfaitement aux exigences du projet a été choisi.

L'intégration de la musique a été réalisée grâce aux outils de création de jeux Godot. Un nœud dédié à la diffusion de la musique a été créé et configuré pour le menu d'accueil. Cette implémentation permet une lecture fluide et immersive de la musique, contribuant ainsi à l'atmosphère du jeu.

```
_audio = GetNode("AudioStreamPlayer") as AudioStreamPlayer;
_audio!.Stream = (AudioStream)GD.Load("res://SFX/music_menu.wav");
_audio.Playing = true;
```

Afin d'offrir une meilleure expérience utilisateur, une barre de son a été ajoutée au menu d'accueil. Cette fonctionnalité permet aux joueurs de régler le volume de la musique à leur convenance, contribuant ainsi à un confort d'écoute optimal.

```
private void _on_son_scroll_bar_value_changed(double value)
{
        _audio.VolumeDb = (float)value-100;
}
```

L'intégration de la musique dans Unununium s'est avéré un succès. L'objectif initial de 10% de complétude a été atteint, et les fonctionnalités implémentées contribuent à l'immersion et au confort des joueurs. La barre de son offre une flexibilité supplémentaire et permet une personnalisation de l'expérience auditive.

3.8 Le Site Web

La création du site web d'Unununium a débuté par un défi. Il s'agissait de trouver un style visuel qui puisse satisfaire l'ensemble de l'équipe. Pour ce faire, nous avons utilisé l'outil de design collaboratif Figma. Cet outil nous a permis de créer une maquette interactive et de tester différentes options de design avant de prendre une décision finale.

Figma nous a permis de créer une maquette détaillée et interactive de notre site web. Grâce à ses nombreux modules intégrés, nous avons pu exporter une grande partie du contenu en HTML/CSS, incluant les pages et leurs composants. Pour dynamiser le site et le rendre plus interactif, nous avons intégré des interactions aux boutons et ajouté des images illustrant notre projet. Durant tout le processus de création, nous avons mis à jour régulièrement les bibliothèques utilisées. Cette démarche vise à garantir la transparence envers les joueurs et à reconnaître le travail des créateurs qui partagent leurs ressources.

Le site web final est accessible via le lien suivant : https ://unununium-company.github.io/unununium-html/index.html ou via notre QR code :

L'utilisation de Figma et la collaboration de l'équipe ont été essentielles à la création d'un site web fonctionnel, esthétique et accessible. Le site web d'Unununium est un outil précieux pour présenter notre projet et interagir avec les joueurs.

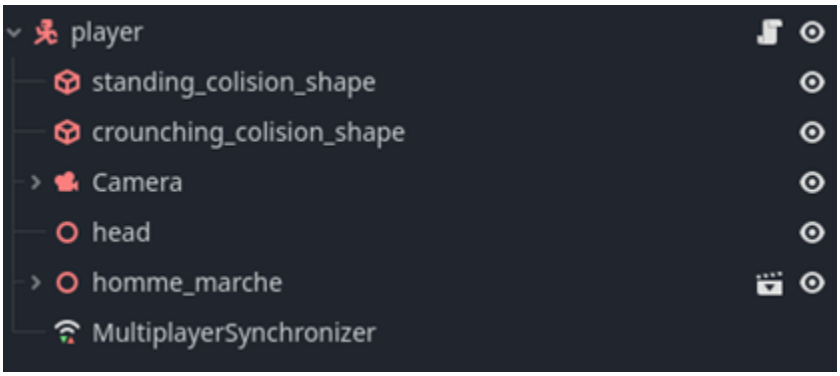
4 Les problèmes rencontrés & les solutions apportées

4.1 Le Gameplay

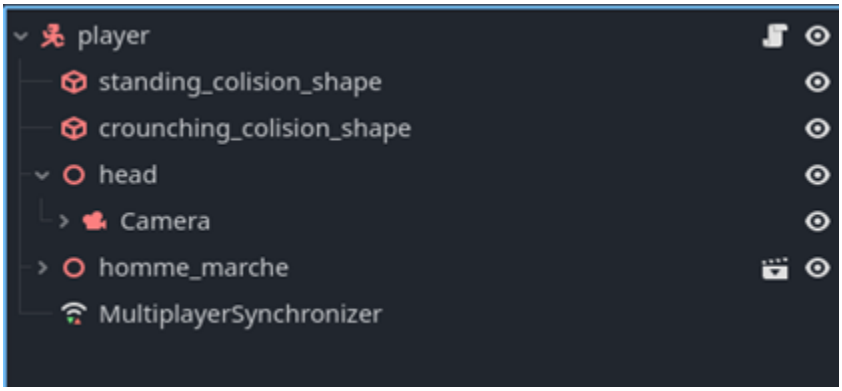
Durant la réalisation de notre jeu vidéo, nous avons été confrontés à de nombreux problèmes.

Concernant le joueur, des problèmes de synchronisation de camera et de déplacement ont été constatés dès le début.

Lorsque le joueur se déplaçait à l'aide des touches, le personnage se déplaçait. En revanche, si le joueur tournait la caméra, le personnage ne se tournait pas, ce qui impliquait que le personnage ne changeait pas de direction. Pour remédier à ce problème, nous avons intégré la caméra à la tête. De ce fait, la caméra est liée directement au corps sans pour autant devoir synchroniser manuellement les rotations de la caméra au corps.
Nœuds du personnage avant modification :



Nœuds du personnage après modification :



Lors de la création du multijoueur, nous avons constaté un souci de proportion. La caméra n'était pas au niveau de la tête comme nous le pensions.

Cette image illustre notre propos. On remarque que le curseur qui représente la vision de l'un des joueurs ne se trouve pas au niveau du visage de l'autre joueur. Pour rectifier ce décalage, nous avons changé les proportions du model 3D en le réduisant à 70% de sa taille originale.

On peut maintenant constater sur cette seconde image, que les personnages ont bien leur visage au même niveau.

Néanmoins, nous avons rencontré un ultime problème, l'un des plus complexes à régler celui de la « position accroupie » du personnage. La solution paraissait pourtant simple, baisser la caméra et réduire la taille du corps. Or, il fallait prendre en compte que le personnage puisse se déplacer tout en restant accroupi. Nous avons donc décidé que la vitesse serait réduite, dès que le personnage s'accroupirait. Puis nous avons seulement baisser la position de la caméra grâce à la position de la tête.

Compte-tenu du temps imparti et du nombre de tâches encore à réaliser, nous avons jugé plus pertinent de définir deux corps, l'un en station « debout » et l'autre, en station « accroupie ». Nous permettant de changer la visibilité de ces statuts grâce à des booléens.

En témoigne cet extrait de code :

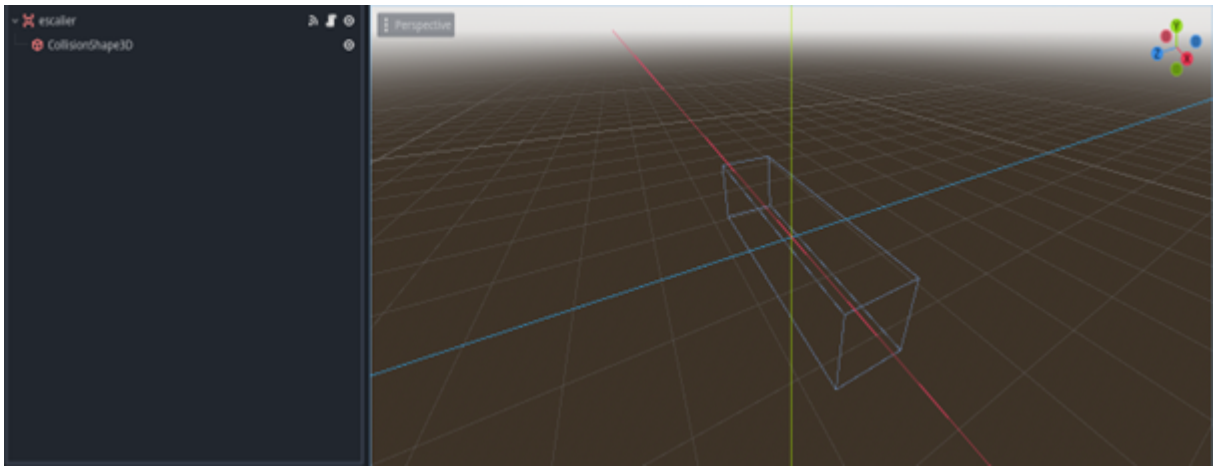
```
// si appuie touche accroupie changement de vitesse et de position de la camera
//et du colision shape
moveSpeed = crouching_speed;
head.Position = new Vector3(head.Position.X, 1.8f + crouching_depth,
        (float)delta * lerp_speed);
standing_colision_shape.Disabled = true;
crounching_colision_shape.Disabled = false;
```

Nous avons aussi été confrontés à de nombreux problèmes lors de la création de la maison. La maison a été créée dans Blender (Wikipédia : Blender est un logiciel libre de modélisation, d'animation par ordinateur et de rendu en 3D, créé en 1994), générant certains problèmes de mis à l'échelle. En effet, le modèle 3D du personnage passait au travers de certains murs.

Ces difficultés étant résolues, il n'en restait pas moins le problème des collisions constaté lors de la deuxième importation du modèle 3D de la maison. Par conséquent, les personnes chargées de la maison ont dû refaire un modèle 3D avec moins de détails pour faciliter la création des collisions automatiques gérées par Godot.

La problématique de la montée des escaliers s'est ajoutée à toutes ces difficultés. Le personnage était dans l'incapacité de monter les escaliers. Nous avons testé de nombreuses méthodes mises à notre disposition par les utilisateurs de Godot sur discord, YouTube, X (anciennement Tweeter), ou même Instagram. En vain, sans grande réussite, une véritable déception pour l'équipe. Une idée a émergé. Nous devions recréer chacune des marches, une par une. C'est une idée audacieuse et peu optimisée par les ressources disponibles, nous viserons à y remédier à l'avenir. Vous pouvez vous poser la question du comment. Nous y répondons plus loin dans notre rapport.

Pour répondre aux problèmes les plus urgents, nous avons créé un modèle de marche qui sert de collision individuelle à chaque marche. Le joueur entre dans l'espace délimité par cette instance de « marche ». Le personnage qu'il contrôle, modifie son axe des Y pour monter grâce à une fonction. Cette fonction est créée dans cet optique.
Ci-dessous une image du bloc de collision qui sert de référence à chaque marche.



Nous avons associé à ce bloc le code suivant :

```
private void _on_escalier_child_entered_tree(Node node)
{
    if (node.HasMethod("Escalier"))
    {
        node.Call("Escalier");
    }
}
```

Etant donné que le bloc en charge de détecter les collisions est un enfant du bloc, cette fonction se charge de détecter le joueur et d'appeler la fonction associer au « Player ».

Nous avons associé à ce bloc le code suivant :

```
public void Escalier()
// permet de monter les escalier au contacte des marches
{
  {
      velocity.Y = 2.0f;
      Velocity = velocity;
      MoveAndSlide();
 }
}
```

La fonction ci-dessus augmente l'axe des Y et donc permet de monter chaque marche.

Cette méthode présente de nombreux désavantages. Elle n'est pas optimisée. En effet, si nous devions coder des centaines de marche, il aurait été trop long de dupliquer le modèle pour chaque marche.

Quand il descend, au contact du bloc, le joueur subit l'effet du changement des axe des Y car il est augmenté à chaque contact. Ceci n'empêche pas la descente mais cela ralenti fortement le joueur. Le but à atteindre est de pouvoir monter et descendre les escaliers. C'est pourquoi nous avons choisi cette option afin de poursuivre notre action. Nous pourrons l'améliorer par la suite.

4.2 Le Game design

Le game design à poser plusieurs problèmes. Tout d'abord, sur le choix des types de nœud pour les interactions, on aurait pu mettre en place un RayCast3D. Une sorte de laser émis par le joueur pour détecter où il regarde et s'il y a des collisions face à lui (is_colliding) avec ses collisions. On pourrait donc récupérer les propriétés de l'objet grâce à un get_collider comme son nom l'indique ou une interaction donnée à l'objet pour ensuite les afficher au joueur grâce à un Prompt (un Label sous le nœud récupéré dans le script) avec une exception pour ne pas détecter le joueur lui-même.

Pour l'interaction, il aurait donc fallu récupérer le moment où la touche est activée lorsque le joueur regarde l'objet en question pour ensuite récupérer son string et activer son interaction. Une autre façon pour les interactions aurait pu être la création de deux classes Interactable et Interactor Interactable récupérer des signaux Focused Unfocused et Interact.
Dans la classe Interactor, on définit les signaux et si l'instance et valide, nous ajoutons une fonction qui récupère l'objet avec qui nous pouvons interagir la plus proche grâce à un get_overlapping_areas et une liste qu'on va parcourir et détecter s'il y a une area avec laquelle on peut interagir. On peut ensuite récupérer leurs positions globales et avec une condition : vérifier quelles interactions est la plus proche.

Puis, on crée un nœud qui hérite de Interactor qu'on nomme PlayerInteractor et qui récupère les propriétés du joueur. Un _physic_Process qui va donc être appelé toutes les frames va vérifier si le joueur est proche ou non d'un objet avec lequel il peut interagir et si le joueur n'est pas assez proche, il envoie le signal unfocus.

Une méthode input sera utilisée pour vérifier si le joueur appuie sur la touche "interact" et si l'instance est valide, il va donc envoyer le signal interact.

Cependant, ces méthodes étaient complexes à mettre en place, faisable en Godot, mais plus compliqué lorsqu'on les passe en C# que ce soit leurs codes, leurs références et lorsqu'on va faire hériter les objets de ces classes.

On a donc préféré rester sur une approche simple du code pour les mécanismes. Pour qu'ils soient facilement compréhensibles pour pouvoir mieux les débugger et les utiliser.

4.3 Les Graphismes

La hitbox générée par Godot pour le sol de la map présentait des irrégularités. Cela provoquait des problèmes de collision, le personnage pouvant traverser le sol à certains endroits.
La géométrie du sol était trop complexe, ce qui rendait la génération de la hitbox difficile et contribuait aux problèmes de collision.

Le sol a été remodelé de manière plus simple et plus propre. Cela a permis de corriger les problèmes de collision et d'améliorer la performance du jeu.
Un environnement extérieur a été ajouté à la map, ce qui n'était pas prévu dans l'idée originale. Cela permet d'offrir une expérience de jeu plus immersive et cohérente.

En outre, les premiers tests ont permis de déceler des problèmes de collision liés à la hitbox du sol. Ces problèmes ont été corrigés en remodelant le sol de manière plus simple. Un environnement extérieur a également été ajouté à la map.

Des tests supplémentaires seront nécessaires pour valider les modifications apportées à la map. La prochaine étape consistera à implémenter les éléments de gameplay et à peaufiner l'environnement.

4.4 Les Menus

La création des menus et l'implémentation des boutons et des entrées de texte associés peuvent s'avérer être des tâches complexes, surtout pour les développeurs débutants. Dans le cadre de notre projet, nous avons rencontré plusieurs difficultés que nous avons dû surmonter pour offrir une expérience utilisateur optimale.

L'une des principales difficultés concerne la gestion de la sortie du jeu. Nous souhaitons implémenter un bouton "Quitter" dans le menu pause qui permettrait au joueur de quitter la partie en cours sans fermer l'application entière. La mise en place de cette fonctionnalité sera bien plus complexe que prévu, car elle nécessite de gérer plusieurs cas de figure :

— Quitter la partie en cours et retourner au menu principal.
— Quitter la partie en cours et revenir au bureau.

Actuellement, notre bouton "Quitter" ferme la fenêtre du jeu. Ce qui provoque l'arrêt total du jeu.

De plus, nous avons rencontré des problèmes lors de la relance d'une partie après en avoir détruit une précédente. Le bouton "Recommencer" présent dans le menu pause ne fonctionnait pas correctement et pouvait entraîner des erreurs. Pour garantir une meilleure stabilité, nous avons décidé de supprimer ce bouton temporairement et d'implémenter la touche F5 qui nous envoie directement au menu game over pour relancer la partie.

Pour résoudre les problèmes rencontrés, nous avons identifié plusieurs solutions possibles :

Gestion de la sortie du jeu :

— Implémenter un système de confirmation avant de quitter la partie en cours, afin d'éviter les actions accidentelles.
— Proposer différentes options de sortie, permettant au joueur de quitter la partie, et donc de revenir au menu principal ou de quitter l'application entière.
— Sauvegarder automatiquement la progression du joueur à intervalles réguliers, dans le but de ne pas perdre les données en cas de sortie inopinée.

La création des menus et l'implémentation des boutons constituent une étape importante du développement d'un jeu vidéo. En rencontrant et en surmontant les défis liés à cette tâche, nous avons acquis une meilleure compréhension des aspects techniques et ergonomiques du design d'interface utilisateur. Les solutions que nous mettrons en place permettront d'améliorer l'expérience de jeu et de garantir une meilleure satisfaction des joueurs.

La gestion des déconnexions s'est avérée être un défi crucial. En tirant parti des signaux intégrés à Godot, nous avons pu les détecter et implémenter des fonctions personnalisées. En cas de perte de connexion, le joueur est redirigé vers le menu principal, avec une réinitialisation complète des outils multijoueurs et de la liste des participants.

Ci-après, nous vous livrons un extrait de code illustrant la mise en œuvre de la solution retenue pour corriger le problème.

```
public override void _Ready()
{
    (...)
    Multiplayer.PeerConnected += PeerConnected;
    Multiplayer.PeerDisconnected += PeerDisconnected;
    Multiplayer.ConnectedToServer += ConnectedToServer;
    Multiplayer.ConnectionFailed += ConnectionFailed;
}


private void PeerDisconnected(long id)
{
   GD.Print("Peer Disconnected  "+id);
   Rpc("Back");
   Back();
}
```

```
[Rpc(MultiplayerApi.RpcMode.AnyPeer)]
private void Back()
{
    GetTree().Paused = false;
    while (Multi_game_manager.Players.Count>0)
    {
        Multi_game_manager.Players.RemoveAt(0);
    }
    var scene =
    ResourceLoader.Load<PackedScene>("res://Gameplay_les_differentes_scenes/menu_


    if (_peer != null)
    {
        if (Multiplayer.IsServer())
        {
            _peer.Host.Destroy();
        }


       _peer.Close();
       _peer.Free();
        Multiplayer.MultiplayerPeer.Close();
        Multiplayer.MultiplayerPeer.Free();
    }
    GetTree().Root.AddChild(scene);
    this.QueueFree();
}
```

4.5 L'Intelligence Artificielle

Le premier problème que nous avons rencontré lors de la création de l'IA est le peu de documentation que l'on peut trouver en C#. Cela nous a donc demandé un peu de temps d'adaptation pour pouvoir comprendre le GDScript et le convertir en C#.

Nous avons ensuite rencontré l'erreur suivante :

Elle se produisait lorsque nous utilisions le nœud NavigationAgent3D du fantôme avant que la carte ne soit synchronisée. Pour résoudre cela, nous avons ajouté dans la fonction _Ready() de l'IA un CallDefered() qui appelle la fonction WaitNavMapSync() qui va attendre que la première image soit générée pour laisser le temps au NavigationServer de se synchroniser avant d'utiliser le NavigationAgent3D.

```
Callable.From(WaitNavMapSync).CallDeferred();

private async void WaitNavMapSync()
{
  await ToSignal(GetTree(), SceneTree.SignalName.PhysicsFrame);
  _navAgent.TargetPosition = _player.Position;
}
```

Pour l'implémentation de l'interaction entre IA et joueur, utiliser les signaux ne fut pas quelque chose d'aisé. De plus, pour connecter les signaux, il faut le faire depuis le nœud dont on veut relier au signal en appelant la fonction Connect() ce qui requiert d'obtenir le nœud qui envoie le signal. Et pour ce faire, il faut donc que le nœud qui souhaite récupérer l'autre soit instancié après.
Le problème est que dans notre cas l'IA a besoin du nœud du joueur pour le pathfinding

```
_player = GetNode<CharacterBody3D>('/root/Node3D/1');
_navAgent.TargetPosition = _player.Position;
```

Le joueur a besoin du nœud de l'IA pour se connecter au signal Caught de l'IA indiquant que le joueur s'est fait toucher.

Script du joueur :

```
private void ConnectSignalToAI()
{
    //Connect le signal Caught de fantome.cs
    GetNode<CharacterBody3D>("/root/Node3D/fantome").Con
                nect("Caught", new Callable(this, "_onEnemyCaught"));
}
```

Cependant, il est impossible d'instancier l'IA et le joueur en même temps. Ainsi, il y aura obligatoirement un nœud qui sera instancié avant l'autre, ce qui pose comme problème que le premier nœud instancié ne pourra pas récupérer l'autre nœud, car cela générera une erreur. Pour remédier à cela, étant donné que c'est le script de la scène principale qui instancie l'IA et le joueur, elle sera instanciée avant eux. Nous avons donc rajouté dans la fonction _Ready() de ce script un signal envoyé au script du joueur pour lui dire que l'IA a bien été instanciée (car dans notre cas, c'est le joueur qui est instancié en premier) et donc que le script du joueur peut récupérer le nœud de l'IA.

Script de la scène principale :

```
    EmitSignal(SignalName.AIReady);
    // Connecte a la fonction ConnectSignalToAI() du joueur
```

Script du joueur :

```
    _private void ConnectSignalToAI()
{
    //Connecte le signal Caught de fantome.cs
    //GD.Print("Enemy signal connected to Player");
    GetNode<CharacterBody3D>("/root/Node3D/fantome").Con
                nect("Caught", new Callable(this, "_onEnemyCaught"));
}
```

4.6 Le Multijoueur & Réseau

Durant les premiers tests de notre application, nous avons observé un comportement inattendu : le joueur était capable de contrôler simultanément les deux personnages.

Afin de garantir un contrôle individuel de chaque personnage, il était nécessaire de dissocier l'exécution du code. Pour ce faire, nous avons intégré les lignes de code suivantes :

```
public override void _Ready()
{
 GetNode<MultiplayerSynchronizer>("MultiplayerSynchronizer").Set
        MultiplayerAuthority(Id);        //defini priorite des id
  (...)
}

public override void _Input(InputEvent ev)
{
   if (GetNode<MultiplayerSynchronizer>("MultiplayerSynchronizer").Get
        MultiplayerAuthority() ==
            Multiplayer.GetUniqueId())
            //execute que sur le bon joueur
   {
        (action des touches)
   }
}
```

Lors de la deuxième phase de tests, nous avons identifié un problème de synchronisation entre les joueurs.

La raison était qu'il fallait nommer le nœud des joueurs en fonction de leur identifiant, c'est pour cela que l'on a fait la liste des joueurs lors du lancement de la partie.

Le code ci-dessous instancie tous les joueurs présents dans la liste précédemment créée, en utilisant leur identifiant unique comme clé de différenciation.

En ce qui concerne les points d'apparition, deux ont été prévus pour la version actuelle du jeu. Cependant, l'architecture du code permet une extension aisée pour en ajouter davantage si le nombre de joueurs venait d'augmenter.

```
public override void _Ready()
{
    foreach (var player in Multi_game_manager.Players)
    {
        player currentPlayer = _playerScene.Instantiate<player>();
        currentPlayer.Id = player.Id;
        currentPlayer.Name = player.Id + "";
        AddChild(currentPlayer);
        if (player.Id == 1 )
        {
            currentPlayer.GlobalPosition =
                GetNode<Node3D>("SpawnPoint/0").GlobalPosition;
        }
        else
        {
            currentPlayer.GlobalPosition =
                GetNode<Node3D>("SpawnPoint/1").GlobalPosition;
        }
    }
```

4.7 Le Son

Le processus de création de la barre de son s'est déroulé sans heurts, et la diffusion de la musique s'est effectuée sans aucune erreur.

4.8 Le Site Web

Dès le début du projet, nous avons pris la décision stratégique de confier la création du site web aux membres de l'équipe ayant les compétences les plus élevées en HTML/CSS.

Cette décision visait à garantir un développement fluide et efficace, en minimisant les risques de problèmes et en maximisant la capacité à les résoudre rapidement.

Le choix de s'appuyer sur les compétences internes en HTML/CSS s'est avéré judicieux pour plusieurs raisons.

Les membres de l'équipe sélectionnés possédaient une connaissance approfondie des langages HTML et CSS, et d'une expérience pratique dans la création de sites web.

Le recours à des ressources externes pour le développement du site web aurait impliqué des coûts supplémentaires et aurait pu générer des problèmes de communication et de coordination.

Le choix de confier la création du site web aux membres de l'équipe a eu un impact positif sur le projet.

Grâce aux compétences et à l'expérience des développeurs internes, le site web a été développé sans rencontrer de problèmes majeurs qui auraient pu retarder ou bloquer l'avancement du projet.

La communication fluide et la collaboration étroite entre les équipes ont permis de gagner du temps et d'améliorer l'efficacité du développement. Le développement interne du site web a permis de garantir une meilleure cohérence avec l'image et les objectifs du projet, et a contribué à la satisfaction des équipes.

En conclusion, la décision de confier la création du site web aux membres de l'équipe ayant les compétences les plus élevées en HTML/CSS s'est avérée un choix stratégique judicieux. Cette décision a permis de garantir un développement fluide et efficace, sans obstacles majeurs, et a contribué à la réussite du projet.

5 Les attentes pour la prochaines soutenances

5.1 Le Gameplay

Cette partie du rapport présente les attentes du groupe pour la prochaine soutenance de notre projet de jeu vidéo du point de vue du gameplay.
Elle s'articule autour de quatre points clés : la compatibilité multi-plateforme, la personnalisation des touches, l'intégration d'animations 3D, l'optimisation des déplacements et des temps de chargement.

Compatibilité multi-plateforme :
Notre objectif est que le jeu soit accessible au plus grand nombre de joueurs possible. C'est pourquoi nous attendons de la prochaine soutenance la validation de la compatibilité du jeu sur les trois systèmes d'exploitation principaux : Linux, Windows et MacOS.

Personnalisation des touches :
Afin d'offrir une expérience de jeu optimale et immersive, nous souhaitons que les joueurs puissent personnaliser les touches de leur clavier. La possibilité de jouer en "QWERTY" est également importante pour certains utilisateurs. La validation de cette fonctionnalité est donc attendue lors de la prochaine soutenance.

Intégration d'animations 3D :
Pour enrichir l'expérience de jeu et la rendre plus fluide, nous avons intégré des animations 3D pour chaque état du personnage (course, marche, accroupissement). La validation de ces animations lors de la soutenance est essentielle pour garantir une expérience utilisateur optimale, notamment en mode multijoueur.

Optimisation des déplacements et des temps de chargement :
Deux points importants pour la fluidité du jeu et le confort des joueurs sont l'optimisation des déplacements du personnage, notamment dans les escaliers, et la réduction des temps de chargement. Nous attendons de la prochaine soutenance des solutions concrètes pour améliorer ces aspects du jeu.

En conclusion, les attentes du groupe pour la prochaine soutenance se concentrent sur la validation de la compatibilité multi-plateforme, de la personnalisation des touches, des animations 3D et de l'optimisation des déplacements et des temps de chargement. La réalisation de ces points permettra de proposer un jeu de qualité et d'offrir une expérience utilisateur optimale aux joueurs.

5.2 Le Game design

Pour le futur, nous aimerions rajouter d'autres interactions qui vont rendre le jeu plus intéressant et varié pour permettre différentes approches. Par exemple, verrouiller une porte de nous-mêmes pour empêcher le fantôme de connaître votre position, ou d'encore récupérer un code en faisant des tâches qui vont nous permettre de déverrouiller un cadenas, ce qui sera une alternative aux clés.

Avec différents objets, pour les récompenses, pour avoir plus de variétés, on aimerait rajouter des traces de l'histoire de la maison ou du fantôme. Cela ajouterait du contenu au Game Design et à l'histoire de notre jeu. Ces traces seront des livres, des photos, qui permettent au journaliste d'avancer dans sa quête et de repartir avec son objectif atteint. Nous voulons aussi meubler la maison pour pouvoir éviter le fantôme et donner de l'identité à la maison.

On pourra donc ajouter des cachettes qui nous rendront indétectables aux yeux du fantôme, que ce soit de se mettre en dessous d'un lit, d'une table ou de se cacher dans une armoire. Pour rendre le jeu plus simple. On pourra mettre une trappe qui donnera à la cave d'une manière différente que l'escalier et qui permettra d'esquiver le fantôme dans un dernier élan.

5.3 Les Graphismes

Le développement du projet n'a pas encore permis la création de textures et graphismes originaux. Des ressources gratuites en ligne ont été importées pour combler ce manque temporaire.
L'importation des UV pour la map a rencontré des difficultés, impactant la qualité visuelle.

Il nous faut intégrer les nouveaux assets au fur et à mesure de leur finalisation.
Identifier les causes des problèmes rencontrés lors de l'importation des UV.
Explorer des solutions techniques alternatives pour optimiser l'alignement des textures.
Tester et implémenter la solution la plus efficace pour garantir une meilleure qualité visuelle de la map.

L'optimisation des textures et graphismes est un élément crucial pour l'aboutissement du projet. La mise en place d'un plan d'action concret et l'allocation des ressources adéquates permettront d'améliorer considérablement la qualité visuelle et l'expérience utilisateur.

5.4 Les Menus

Chez Unununium, nous sommes convaincus que chaque joueur est unique et mérite une expérience de jeu sur mesure. C'est pourquoi nous sommes fiers de vous annoncer l'implémentation prochaine d'un menu de configuration des contrôles complet, accessible avant la sortie du jeu.

Le menu "Configuration des contrôles" sera abordable depuis le menu principal et pause, via un bouton dédié "Paramètres des contrôles". Ce menu clair et intuitif vous permettra de modifier l'ensemble des contrôles du jeu, selon vos préférences et votre style de jeu.

De nombreuses options de personnalisation seront disponibles. Vous pourrez modifier chaque action du jeu à la touche ou au bouton de votre choix, sur votre clavier. Assignez les actions les plus importantes aux touches les plus accessibles pour maximiser votre réactivité et votre efficacité en jeu.

Le menu "Configuration des contrôles" est conçu pour vous simplifier la vie. Si vous attribuez accidentellement la même touche à deux actions différentes, le menu vous avertira de l'erreur et vous proposera de corriger le problème. Vous pourrez ainsi jouer en toute confiance, sans vous soucier de conflits de touches malencontreux.

La possibilité de remapper entièrement les contrôles est un élément essentiel pour rendre notre jeu ouvert à tous les joueurs. Nous sommes convaincus que cette fonctionnalité permettra à chacun de profiter pleinement de l'expérience Unununium et de vivre des aventures inoubliables.

5.5 L'Intelligence Artificielle

Afin de dynamiser l'expérience utilisateur et d'accroître la difficulté du jeu, les modifications suivantes seront apportées au prototype pour la prochaine soutenance :

Intégration d'un "screamer" : Un effet sonore strident sera déclenché lorsque l'IA touche le joueur, contribuant à l'immersion et à la tension du joueur.
Augmentation progressive de la vitesse de l'IA : A chaque énigme résolue, la vitesse de l'IA augmentera, créant un sentiment d'urgence et incitant le joueur à progresser rapidement.

Le développement de l'IA est quasiment achevé. Seuls quelques détails mineurs restent à peaufiner avant la prochaine soutenance.

Intégration du screamer : Définir le type de son, le moment précis de son déclenchement et son volume pour garantir une expérience optimale.
Ajustement de la vitesse de l'IA : Déterminer la courbe de progression de la vitesse pour maintenir un niveau de difficulté stimulant et équilibré.
Test et validation : Tester les modifications apportées et recueillir les avis des utilisateurs pour identifier d'éventuels points d'amélioration.

Les modifications prévues pour la prochaine soutenance visent à enrichir l'expérience de jeu et à renforcer l'interaction entre le joueur et l'IA. Le développement de l'IA est en bonne voie et sa finalisation est imminente.

5.6 Le Multijoueur & Réseau

Le développement du mode multijoueur a permis de mettre en lumière plusieurs points d'amélioration :

Gestion des erreurs en mode solo après une session multijoueur :

Des erreurs surviennent lors du lancement d'une partie solo après avoir joué en mode "Host" ou "Join". La cause réside dans l'incapacité de Godot à attribuer un ID à un second joueur inexistant en mode solo.

Le deuxième problème à régler est l'optimisation de la bande passante. En effet, le multijoueur actuel utilise une quantité importante de bande passante pour synchroniser les positions du joueur et du fantôme. Pour réduire la consommation de bande passante, il est possible de diminuer la fréquence des synchronisations. Cependant, cela peut générer des mouvements saccadés et une expérience de jeu moins fluide. Une solution consiste à implémenter une fonction de lissage des mouvements afin de compenser la réduction de la fréquence des synchronisations.

Synchronisation des interactions :

Le mode multijoueur est fonctionnel, mais la synchronisation d'interactions telles que l'ouverture de portes, la résolution d'énigmes ou l'utilisation de médicaments n'est pas encore implémentée. Le développement de ces fonctionnalités est prévu pour la prochaine soutenance.

Le mode multijoueur est une fonctionnalité importante du jeu et son développement est en bonne voie. Les points d'amélioration identifiés seront traités afin d'offrir une expérience de jeu optimale aux joueurs.

5.7 Le Son

Dans le cadre de notre prochaine soutenance, nous avons pour ambition d'enrichir l'expérience utilisateur de notre jeu vidéo en y intégrant une bande-son immersive et des effets sonores pertinents. Nous attribuerons une musique de fond à chaque menu afin de créer une ambiance en accord avec le jeu. Nous intégrerons des effets sonores associés aux actions du joueur, telles que la course, la marche ou la récupération d'objets, pour renforcer l'immersion et le dynamisme du gameplay. En somme, nous finirons par ajouté une musique de poursuite lors de la traque du joueur par les ennemis, accentuant le suspense et l'intensité de l'expérience.

5.8 Le Site Web

Dès le début de ce projet, nous avons placé la barre haut pour le site web. Notre ambition est de le doter de toutes les fonctionnalités nécessaires pour le rendre à la fois complet et représentatif de notre projet.

En termes de fonctionnalité, le site web offrira aux utilisateurs :
- Une présentation claire et concise du projet, incluant ses objectifs, ses ambitions et son équipe.
- Un accès facile aux informations essentielles, telles que les actualités, les événements, les captures d'écran et les vidéos.
- Une navigation fluide et intuitive sur tous les supports (ordinateurs, tablettes, smartphones).

L'aspect visuel du site web sera tout aussi important que sa fonctionnalité. Nous visons un design :
- Moderne et attractif, qui reflète l'univers et l'ambition du projet.
- Soigné et professionnel, pour inspirer confiance et crédibilité aux utilisateurs.
- En cohérence avec l'identité visuelle du projet, pour une expérience homogène.

Le site web ne sera pas figé dans le temps. Nous le concevons comme un outil évolutif qui s'enrichira au fil du projet :
- Ajout de contenus réguliers (actualités, articles, tutoriels, etc.) pour maintenir l'intérêt des utilisateurs.

En conclusion, le site web est bien plus qu'une simple vitrine pour notre projet. Il est un outil essentiel pour :

— Communiquer avec les joueurs et les parties prenantes.
— Promouvoir le projet et le faire connaître à un large public.
— Créer une communauté autour du projet et fédérer les participants.

Nous sommes conscients que la réalisation d'un site web complet et représentatif de notre projet est un défi ambitieux. Cependant, nous sommes déterminés à mettre tout en œuvre pour atteindre cet objectif et offrir aux utilisateurs une expérience optimale.

6 Conclusion

Au cours de ces dernières semaines, Unununium a rencontré quelques difficultés dans chaque partie du développement du jeu telle que : les énigmes, ou les collisions entre le joueur et son environnement. Malgré tout, l'équipe a réussi à debugger le code et surmonter ces problèmes afin de proposer une première version de "Les maisons aux âmes perdues" qualitative le plus rapidement possible. D'ici le prochain rapport, il reste certaines tâches à accomplir afin de terminer le jeu et permettre à ses joueurs une expérience inédite et complète. Ces tâches sont les suivantes :

1. Compléter les énigmes
2. Ajouter les musiques et effets sonores
3. Améliorer les déplacements de l'IA
4. Optimiser les déplacements du joueur
5. Terminer le Site Web

Notre équipe a confiance en ce qui est du respect du planning concernant le premier niveau de ce jeu. Nous avons réussi à atteindre nos objectifs pour la première soutenance. Il se peut que nous ayons un peu d'avance par rapport à nos objectifs, en particulier au niveau des menus.