



RAPPORT  
PROJET DE PROGRAMMATION - SEMESTRE 2

---

## Nyctalopia

---

Créé par :

Iñigo Aldaraborda Hoyos - Chef de projet

Hugo Meleiro

Guilhem Cros

Marin Godefroy

EPITA Toulouse

Mardi 26 Avril 2022

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Répartition des tâches</b>	<b>2</b>
<b>3</b>	<b>Progrès réalisé</b>	<b>3</b>
3.1	I.A. - Intelligence Artificielle . . . . .	3
3.2	Audio et Effets Sonores . . . . .	4
3.3	Gameplay . . . . .	5
3.4	Communication . . . . .	14
3.5	Graphismes, Modèles et Terrain . . . . .	16
3.6	Multijoueur . . . . .	20
3.7	UI/UX - Interface . . . . .	21
3.8	Manuels d'instructions et d'installation . . . . .	23
3.9	Site Web . . . . .	24
<b>4</b>	<b>Planning</b>	<b>28</b>
<b>5</b>	<b>Difficultées</b>	<b>29</b>
<b>6</b>	<b>Conclusion</b>	<b>30</b>
<b>7</b>	<b>Annexe</b>	<b>31</b>

# 1 Introduction

Dans le cadre du projet de S2 à EPITA, nous sommes poussés à mettre en pratique les différentes connaissances acquises en Travaux Pratiques et en cours, dans la réalisation d'un projet en groupe. Le projet de notre studio gameHUB est la création d'un jeu d'horreur : *Nyctalopia*.

À l'arrivée de la première soutenance, nous avons préféré nous concentrer sur l'aspect technique du jeu, le “backend”, toute la partie que le ou les joueurs ne verraient pas, ce qui nous a permis pour cette deuxième date clé de notre projet, d'avancer plus rapidement afin d'obtenir des résultats plus visibles surtout dans les parties graphiques et jouables.

Pour la seconde soutenance, ayant eu moins de temps, par rapport à la première soutenance, nous avions décidé de régler les bogues pouvant être très gênants pour l'utilisateur, ainsi qu'implémenter de quelques fonctionnalités au niveau du Gameplay.

Dans cette dernière ligne droite, nous avons essayé de clôturer notre projet en intégrant toutes les parties manquantes du jeu, notamment le nouveau *Chapitre 3 : Les Égouts*. Même si tous nos objectifs n'ont pas pu être atteints, nous avons voulu présenter une version finale propre et fonctionnelle.

Nous présentons, dans ce rapport de projet, tout le progrès réalisé dans notre projet lors de ce semestre dans la réalisation de notre jeu-vidéo d'horreur : *Nyctalopia*.

## 2 Répartition des tâches

Voici la répartition prévue des tâches entre les étudiants du groupe gameHUB. Celle-ci n'a pas changé depuis le cahier des charges.

Tâches	Iñigo	Hugo	Guilhem	Marin
AI - Intelligence Artificielle			◊	✓
Audio	✓			
Communication & Gameplay			✓	◊
Graphismes & Modèles	✓			
Manuels				✓
Map	◊	✓		
Multijoueur	✓	◊		
Saves			✓	
UI/UX & Réseaux		✓		
VFX		✓		
Website				✓

✓ - Responsable

◊ - Support - Suppléant

## 3 Progrès réalisé

### 3.1 I.A. - Intelligence Artificielle

Pour implémenter une intelligence artificielle, nous avons testé divers algorithmes de "pathfinding" dans un grand nombre de scènes comprenant des obstacles afin de pouvoir implémenter dans notre jeu une entité antagoniste qui suivra le joueur.



FIGURE 1 – *Entité*

Notre entité utilise le pathfinding pour déterminer, dès lors que le joueur rentrer dans son champ d'action, un chemin à suivre pour atteindre le joueur et l'attaquer.

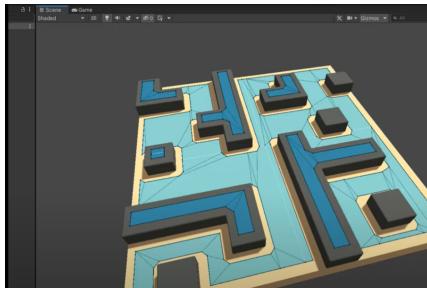


FIGURE 2 – *NavMesh : Tous les endroits possibles au déplacement de l'entité*

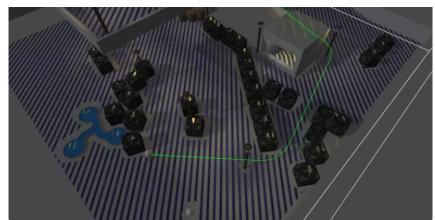


FIGURE 3 – *Pathfinding*

Cet intelligence artificielle est utilisée dans le chapitre 3 du jeu.

## 3.2 Audio et Effets Sonores

La cinématique principale et le menu d'accueil du jeu comptent avec ses propres musiques, libres de droits et utilisables pour usage commercial, qui représentent bien l'aspect mystérieux et effrayant de notre jeu d'horreur.

Le menu d'accueil et l'interface d'utilisateur ont aussi leurs propres effets sonores.

Dans le jeu, le joueur entendra pendant tout le jeu de la musique d'ambiance en arrière-plan correspondant à l'environnement dans lequel il se trouve. Cette musique d'ambiance provenant de *YouTube* est reproduite en boucle et est libre de droits. Des effets sonores circonstanciels peuvent aussi être rencontrés dès que le joueur traverse une zone ou dès lorsqu'il active un des déclencheurs de sons qui sont là pour effrayer le joueur.

Par ailleurs, dès que le joueur rapprochera une borne de sauvegarde, il le saura à cause de la musique qui provient de celle-ci. Pour rendre hommage au jeu vidéo *Fallout 3*, cette chanson est *I Don't Want to Set the World on Fire* de *The Ink Spots*. Il s'agit bien d'une musique libre de droits.



FIGURE 4 – Borne de sauvegarde

### 3.3 Gameplay

Concernant la partie jouable, nous avons tout d'abord appris à nous familiariser avec les différents outils mis à disposition par *Unity3D* pour enregistrer les entrées du joueur, telles que les appuis sur les touches du clavier ou les mouvements de la souris. Nous avons finalement choisi d'utiliser le *New Input System*, qui nous permettrait par la suite de pouvoir implémenter des touches personnalisables, caractéristique que nous n'avons pas eu le temps d'implémenter.

Par la suite nous avons donc crée le script responsable du mouvement de la caméra du joueur, celui-ci permet au joueur de déplacer la caméra dans la limite du physiquement possible par un humain, pour garder la sensation d'un jeu à la première personne dans lequel on incarne un humain. La caméra est donc bloquée à la verticale en haut et en bas, et les mouvements sur le côté tournent le corps du joueur.

Le script permettant au joueur de se déplacer a été le suivant à être implémenté. Il permet au joueur de se déplacer avec quatre touches, lui servant à se déplacer vers l'avant, en arrière, à gauche et à droite. Une touche permettant de s'accroupir, un *RayCast* est utilisé pour savoir si le joueur peut se relever, si un objet ne se situe pas au-dessus de lui. Enfin une touche pour courir a été ajoutée. Nous avons décidé de ne pas implémenter d'action "sauter" pour limiter les déplacements du joueur et rendre le *Gameplay* plus oppressant. Pour les actions "courir" et "s'accroupir", un *CallbackContext.performed* a été utilisé pour ne pas avoir à maintenir la touche enfoncee.

Enfin, pour cette dernière soutenance, nous avons ajouté plusieurs scripts à notre projet :

```
public class PlayerStatut
{
    public int currentSavePoint;
    public Vector3 CurrentSaveCoo => savePoints[currentSavePoint];

    public List<Item> items;
    public List<Env> envStatut;
    public List<Vector3> savePoints;
```

FIGURE 5 – *PlayerStatut*

***PlayerStatut*** - une classe permettant de stocker l'avancée du joueur dans sa partie.

Ce script permet de tenir à jour la réussite des différents puzzles et l'ouverture de différentes portes du jeu à travers la récupération des divers objets nécessaires à leur réussite ou leur ouverture. Ce script garde aussi en mémoire les coordonnées du dernier point de sauvegarde utilisé pour y placer le joueur lors de sa prochaine mort ou du prochain lancement de la partie. Pour réaliser ce script, nous avons eu besoin de créer deux autres classes :

- ***Item*** - qui contient l'identifiant de l'objet
- ***Env*** - qui contient l'identifiant des portes et énigmes résolues par le joueur

*PlayerStatut* stocke les différents éléments de *Item* et *Env* dans des listes. Lors de l'ouverture d'une porte nous vérifions si l'*Item* correspondant à la clé est bien dans le *PlayerStatut* du joueur grâce à l'identifiant de l'objet et nous ajoutons cette porte à la liste d'*Env*. Lors du prochain lancement de la partie, tous les *GameObjet* possédant l'étiquette *Door* serons récupérés et ceux ayant le même identifiant verront leur script attaché à *Interactable* exécuté, les portes seront donc ouvertes comme au moment de la sauvegarde, grâce à une méthode de la classe *SaveManager*. Seulement les portes liées à des énigmes ou ouvertes grâce à une clé seront gardées en mémoires.

```

public class SaveManager : MonoBehaviour
{
    private string key = "AY1MSnkPgyEIDaHyDhomIWJgCMcQKHzRXSabOsFgrzzcepzMO";
    private string filePath;
    private List<string> sceneNames;

    private ToJSON file;

    public PlayerStatut playerStatut = new PlayerStatut();

    // Event function
    private void Start()
    {
        filePath = Application.persistentDataPath + "/playerSave.json";
        file = new ToJSON();
        sceneNames = new List<string>() {"Chapitre 1 & 2", "Chapitre 3"};
    }
}

```

FIGURE 6 – *SaveManager*

**SaveManager** - un script lié aux bornes de sauvegarde disséminées à des endroits prédéfinis dans le jeu. Cette classe possède le *PlayerStatut* actuel, le chemin d'accès pour accéder au fichier de sauvegarde récupéré grâce au *Application.persistentPath* mis à disposition par *Unity3D*. Celle-ci possède aussi le nom des différentes scènes utilisées.

De plus, une instance de la classe *ToJSON* est présente dans la classe, permettant de garder en mémoire les sauvegardes, cette classe sera ensuite sérialisée dans le format JSON (1) et chiffrées grâce au chiffrage XOR (XOR Cipher) (2), empêchant ainsi le joueur de tricher en modifiant sa sauvegarde. De plus, cette classe possède différentes méthodes permettant d'ajouter des *Item* et des *Env* mais aussi de vérifier si un *Item* est possédé ou si un *Env* à été complété. Comme évoqué dans la description de la classe *PlayerStatut*, la méthode *Refresh* recharge la scène de la dernière sauvegarde, donne les items que possédait le joueur lors de la sauvegarde et ouvre les portes gardées en mémoire.

```

public class ToJSON
{
    public string save01 = "";
    public string save02 = "";
    public string save03 = "";
}

```

FIGURE 7 – *ToJSON*

**ToJSON** - Cette classe contient les trois emplacements de sauvegarde sous forme de chaîne de caractères au format JSON pour empêcher tout problème de référence. C'est cette classe qui sera transformée en fichier de sauvegarde.

(1) : Le format JSON (JavaScript Object Notation) est un format couramment utilisé pour représenter de l'information structurée sous forme de texte. Il est facilement lisible par des humains (malgré le manque d'espaces nécessaire pour alléger le fichier).

```
{"currentSave":{"x":50.0,"y":0.0,"z":100.0}, "item":[{"id":101}, {"id":102}], "envStatut":[{"id":201}]} 
```

FIGURE 8 – Fichier JSON

(2) : Le chiffrage XOR utilise l'opérateur binaire XOR (OU-exclusif), voici sa table de vérité :

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

FIGURE 9 – Table de vérité OU-exclusif

Lors du chiffrage nous allons passer à travers une porte logique XOR chaque lettre de notre chaîne de caractères avec la lettre au même indice, le tout modulo la longueur de notre clé, et ainsi reconstruire une chaîne de caractères chiffrée, de même longueur mais inintelligible. Lors du déchiffrement nous allons effectuer les mêmes actions mais avec la chaîne de caractères chiffrée à la place de celle en claire. Pour passer deux lettres à travers la porte XOR on passe en réalité les représentations binaires des codes ASCII (American Standard Code for Information Interchange) de ces lettres, le résultat sera donc une représentation binaire convertie en nombre et retranscrite grâce à la table ASCII. Exemple : Dans cet exemple on souhaite chiffrer le mot "MESSAGE" avec la clé "CLE" :

Lettres	M	E	S	S	A	G	E
Code ASCII	77	69	83	83	65	71	69
Binaire	01001101	01000101	01010011	01010011	01000001	01000111	01000101

Lettres	C	L	E
Code ASCII	67	76	69
Binaire	01000011	01001100	01000101

FIGURE 10 – Conversion des lettres en binaire

Le chiffrement XOR va donc correspondre à ceci :

Message en binaire	01001101	01000101	01010011	01010011	01000001	01000111	01000101
Clé en binaire (répétée si nécessaire)	01000011	01001100	01000101	01000011	01001100	01000101	01000011
Message chiffré en binaire	00001110	00001001	00010110	00010000	00001101	00000010	00000110
Message chiffré en code ASCII	14	9	22	16	13	2	6
Message chiffré en lettre	SO	HT	SYN	DLE	CR	STX	ACK

FIGURE 11 – Conversion des lettres en binaire

Dans cet exemple le message chiffré est composé uniquement de caractères ASCII non imprimables et de caractères de contrôle, utilisés pour contrôler certains périphériques ou donner des indications sur le format du document.

Le chiffrement XOR est basique et le message chiffré peut être facilement décrypté si la clé est trop petite par rapport à la taille du message grâce à une analyse fréquentielle ou dans ce cas d'utilisation (sauvegarde), par une attaque à texte clair connu étant donné que le joueur connaît les noms des objets.

Nous avons choisi de créer une clé de 50 caractères choisis aléatoirement parmi les caractères dits imprimables de la table ASCII. Il est donc ici préférable de se référer aux objets par des nombres plutôt que des noms, nous choisissons de représenter les objets, les portes et énigmes réussis par des identifiants. Pour un souci de lisibilité nous avons aussi ajouté un dictionnaire liant les identifiants des objets, portes et énigmes à leurs noms.

De plus, pour l'instant, la clé de chiffrement est unique pour toutes les sauvegardes et tous les joueurs mais il peut être intéressant d'en générer une nouvelle aléatoirement à chaque création d'une nouvelle partie en retranscrivant ce code en C#.

Nous avons aussi implémenté la touche "utiliser" assignée à la touche "E". Nous utiliserons le système de *RayCast* (qui peut être traduit par "lancé de rayon") et le système *Event* proposé par Unity3D pour découper ce système en deux scripts distincts : *Interactor* et *Interactable*. Ils seront respectivement liés au joueur et aux objets que l'on veut interactifs tels que les bornes de sauvegarde, les puzzles, les clés à récupérer, les portes...

```

public void Interact(InputAction.CallbackContext ctx)
{
    RaycastHit hit;
    if (ctx.canceled && Physics.Raycast( origin:cam.transform.position, direction:cam.transform.forward, out hit, range, interactableLayerMask))
    {
        Debug.Log(hit.collider.name);
        if (hit.collider.GetComponent<Interactable>() != null)
        {
            if (interactable == null || interactable.ID != hit.collider.GetComponent<Interactable>().ID)
            {
                interactable = hit.collider.GetComponent<Interactable>();
                gameObject = hit.collider.gameObject;
            }

            if (gameObject.CompareTag("Item"))
            {
                gameObject.SetActive(false);
                save.ItemCollected(new Item(interactable.ID));
            }
            else if(gameObject.CompareTag("Door"))
            {
                save.EnvCompleted(new Env(interactable.ID));
            }
        }

        interactable.onInteract.Invoke();
    }
}

```

FIGURE 12 – *Interactor*

**Interactor** - Ce script est lié au joueur et permet grâce au *Raycasting* de savoir si le joueur pointe son curseur vers un objet possédant la couche *Interactable* et si celui-ci est assez proche du joueur. Ensuite nous récupérons le *GameObject* et nous lançons la fonction liée à l'objet à travers le script *Interactable*. De plus, si l'objet possède l'étiquette *Item*, celui-ci sera ajouté à l'inventaire du joueur et sera désactivé pour le faire disparaître, d'autre part, si l'objet possède l'étiquette *Door*, cette porte sera gardée en mémoire et sera ouverte lorsque le joueur chargera sa sauvegarde. Cette fonction est appelée quand le joueur appuie sur la touche "E".

```

public class Interactable : MonoBehaviour
{
    public UnityEvent onInteract;    ↗ No methods
    public int ID;    ↗ Unchanged
}

```

FIGURE 13 – *Interactable*

**Interactable** - Ce script est lié à l'objet avec lequel nous voulons interagir, il est composé d'un identifiant et d'un *UnityEvent* permettant d'appeler une fonction.

Par la suite, nous avons implémenté la classe permettant d'activer/désactiver la lampe torche.

```
private GameObject GetSpotlight()
{
    GameObject output = null;
    var currentScene = SceneManager.GetActiveScene();
    if (currentScene.name == "Chapitre 1 & 2")
    {
        output = Chap1;
    }
    else
    {
        output = Chap3;
    }

    return output;
}
```

FIGURE 14 – *GetSpotlight*

**GetSpotlight-** Ce script permet de récupérer le *GameObject* qui représente le *SpotLight* (Point Lumineux) représentant la lumière de la lampe torche, dans les chapitres 1 et 2 l'intensité de la lampe torche est moindre que celle du Chapitre 3, en effet le Chapitre 3 se situant dans un endroit clos il est nécessaire d'augmenter l'intensité pour pouvoir y voir correctement. Pour savoir quel *GameObject* nous devons utiliser, nous utilisons le *SceneManager* mis à disposition par *Unity3D* pour pouvoir connaître la Scène actuellement utilisée.

```

public void TurnTheLightOn(InputAction.CallbackContext ctx)
{
    var light:GameObject = GetSpotlight();
    if (ctx.performed)
    {
        if (light.activeSelf)
        {
            light.SetActive(false);
        }
        else
        {
            light.SetActive(true);
        }
    }
}

```

FIGURE 15 – *TurnTheLightOn*

**TurnTheLightOn-** Ce script permet d'éteindre et d'allumer la lampe torche, l'utilisation du *ctx.performed* permet de basculer d'un état à l'autre avec un simple appui sur la touche prévue à cet effet, le *Unity Input System* permet d'appeler cette fonction lorsque la touche "T" est appuyée.

Nous avons enfin implémenté *DevMod* permettant de lancer le jeu en mode Développeur. Cette fonction permet de charger la scène suivante à celle dans laquelle se trouve le joueur.

### 3.4 Communication

Le logo Nyctalopia, après plusieurs étapes de conception, est écrit en noir avec la police *October Crow* et se présente dans le menu principal du jeu ainsi qu'à la fin de la cinématique de début de jeu.



FIGURE 16 – Police d’écriture choisie pour Nyctalopia

Concernant le logo du studio, que nous avons réussi à le désigner assez rapidement, nous nous sommes inspirés du logo de *GitHub*, comme peut en témoigner le nom de notre studio.



FIGURE 17 – Logo choisi pour GameHub Studio

Enfin, pour le logo du jeu, nous avons eu plus de mal à trouver un logo convenable et esthétique pour pouvoir correspondre à une icône Windows, ou une icône de jeu Steam. Finalement nous avons opté pour un logo simple avec l'initiale du nom du jeu dans un hexagone légèrement stylisé avec un effet de lunettes 3D rouge/bleu.



FIGURE 18 – *Icône du jeu*

### 3.5 Graphismes, Modèles et Terrain

*Nyctalopia* suit un système de chapitres. Lors du parcours de chacun des 3 chapitres créés, le joueur retrouvera une zone caractéristique représentant le chapitre. A savoir : la forêt, la base militaire et les égouts. Lors du début de la partie, le joueur retrouve une cinématique d'une minute trente secondes, où le joueur est introduit au monde de *Nyctalopia* et est mis en contexte de l'histoire qui commence par un accident de voiture.

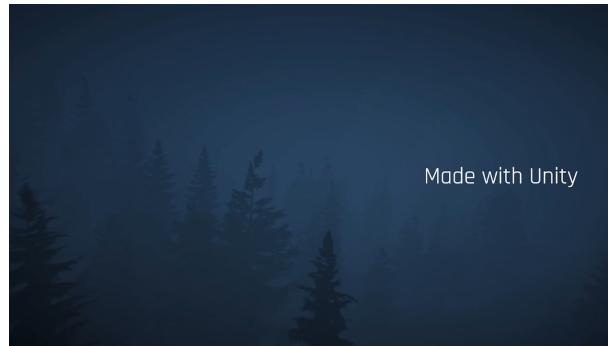


FIGURE 19 – *Prologue*

Le chapitre 1 se passe dans une forêt. Une grande variété de modèles 3D d'arbres, arbustes et herbe ont été sélectionnés et utilisés dans le jeu pour couvrir ce terrain et créer une forêt riche et variée. La plupart de ces modèles de végétation proviennent de la bibliothèque de modèles 3D *Quixel's Megascans* et du site officiel de modèles Unity : *Unity Asset Store*. Plusieurs modifications et conversions ont été nécessaires pour les rendre compatibles avec notre projet qui est basé sur le système de rendu HDRP (*High-Definition Render Pipeline*) qui offre des résultats graphiques plus attractifs.



FIGURE 20 – *Chapitre 1*

Le chapitre 2 se passe dans une base militaire. Une grande variété de modèles 3D d'arbres, arbustes et herbe ont été sélectionnés et utilisés dans le jeu pour couvrir ce terrain et créer une forêt riche et variée. La plupart de ces modèles de végétation proviennent de la bibliothèque de modèles 3D *Quixel's Megascans* et du site officiel de modèles Unity : *Unity Asset Store*. Plusieurs modifications et conversions ont été nécessaires pour les rendre compatibles avec notre projet qui est basé sur le système de rendu HDRP (*High-Definition Render Pipeline*) qui offre des résultats graphiques plus attractifs.



FIGURE 21 – Chapitre 2

Enfin, le chapitre 3, se passe dans des égouts. L'utilisation de modèles 3D modulaires conçus pour la création d'égouts nous ont permis d'avoir le système souterrain désiré.



FIGURE 22 – Chapitre 3 - Les égouts

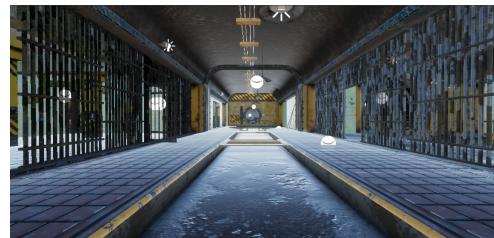


FIGURE 23 – Chapitre 3 - Les égouts

Dans ce chapitre, le joueur retrouve une nouvelle mécanique de jeu : les leviers. Ces leviers après activation, ouvrent des portes dans le chapitre 3. La porte ouverte peut être déduite grâce au son 3D que les portes produisent dès que le joueur active ses leviers correspondants. Les leviers possèdent aussi un indicateur LED qui indique l'état du levier (désactivé, rouge clignotant/activé, vert). Pour reconnaître que le joueur a utilisé la touche *Action* sur eux, ils utilisent le script *Interactable* qui va ensuite lancer le script *LeverPress* mentionnés précédemment.



FIGURE 24 – *Chapitre*

Dans ce chapitre le joueur pourra utiliser aussi une lampe torche pour se guider dans l'obscurité de ce chapitre. Cette lampe torche peut être allumée avec la touche associée à celle-ci (Par défaut, **T**).



FIGURE 25 – *Système de lampe torche*

Concernant les personnages jouables, nous comptons avec deux modèles de personnages complètement animés, un homme et une femme, provenant de la bibliothèque d'Adobe *Mixamo*.



FIGURE 26 – Personnage masculin



FIGURE 27 – Personnage féminin

Pour l'entité, un modèle a déjà été choisi. (c.f. I.A. - Intelligence Artificielle)

### 3.6 Multijoueur

Pour le multijoueur nous avons décidé d'utiliser le SDK Steamworks fourni par Steam Inc. N'étant pas nativement compatible avec C#, nous avons dû utiliser une bibliothèque tierce nommée *Steamworks.NET*. Ce SDK permet de créer des lobbys et d'intégrer une liste d'amis, qui facilitera la communication entre les deux joueurs étant donné que Steam est la plateforme de vente de jeux vidéos la plus populaire au monde avec 100+ millions de connexions mensuelles.

Avec tout ces outils en place, il nous a suffi de lire la documentation officielle de Steam, et de mettre en place un script consistant à créer et à joindre des lobbys grâce aux boutons présents au sein de l'interface (c.f. UI/UX).

Un chat vocal est aussi présent pour les joueurs étant dans une même salle de jeu. Pour cela, on a utilisé *SteamVoice*, une bibliothèque fournie par Steam.

### 3.7 UI/UX - Interface

L'interface utilisateur possède trois parties principales, le menu principal, le menu “Play” et le menu “Paramètres”. Ce dernier permet à l'utilisateur de pouvoir régler la résolution, la taille de la fenêtre (fenêtré, borderless ou plein écran), mais également le son et le choix des touches (AZERTY ou QWERTY). Le menu “Play” possède trois boutons, un placé à gauche, permettant au joueur 1 de créer un lobby et de le rejoindre, et à droite deux boutons permettent au joueur 2 de soit, joindre le lobby avec un *CSteamID* (code de lobby délivré par Steam) ou bien en utilisant sa liste d'ami Steam. Enfin, le menu principal possède deux grands boutons appelant le joueur à soit débuter une nouvelle campagne ou bien de continuer là où il avait sauvegardé pour la dernière fois (c.f. Sauvegardes)

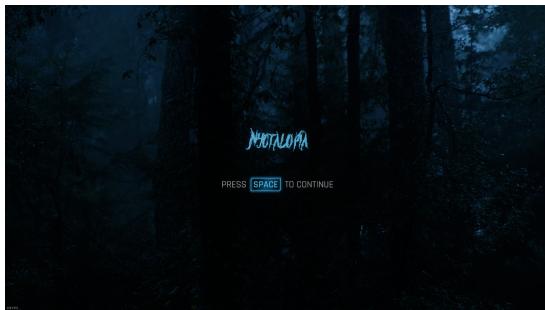


FIGURE 28 – Écran d'accueil

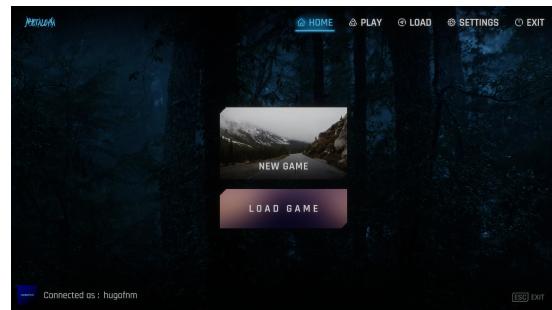


FIGURE 29 – Menu principal

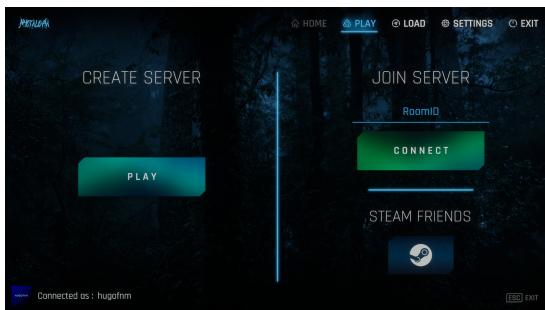


FIGURE 30 – Menu “Play”

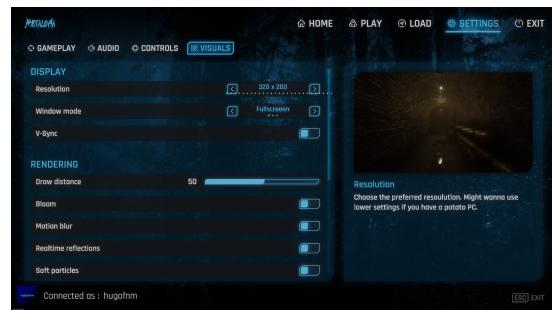


FIGURE 31 – Menu “Paramètres”

Il est désormais possible aussi de changer la langue du jeu. Trois options sont possibles : français, anglais ou espagnol.



FIGURE 32 – Français



FIGURE 33 – *Anglais*



FIGURE 34 – *Espagnol*

Si l'utilisateur n'a pas effectué de choix, la langue par défaut sera réglée par rapport à la langue système (le système d'exploitation envoie au jeu la langue principale du système).



FIGURE 35 – *Sélection de la langue dans le menu Paramètres*

Nous avons également mis en place un installateur graphique “.exe” qui est accessible à tout utilisateur possédant une machine Windows ou Linux 64 Bits sur le site *get.nyctalopia.games* (c.f Site Web). Ainsi qu'un script permettant d'informer ses amis que l'on joue à Nyctalopia sur la plateforme Discord a été créé.

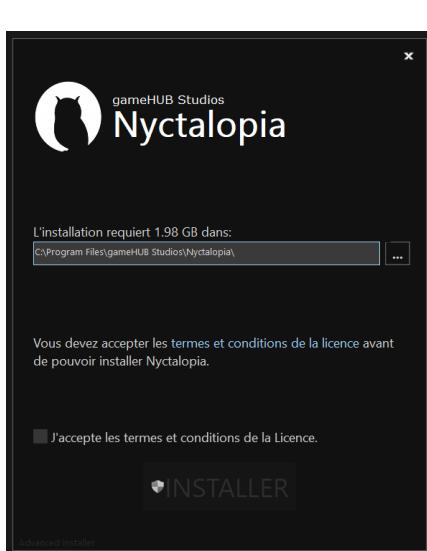


FIGURE 36 – *Installateur Graphique .exe*

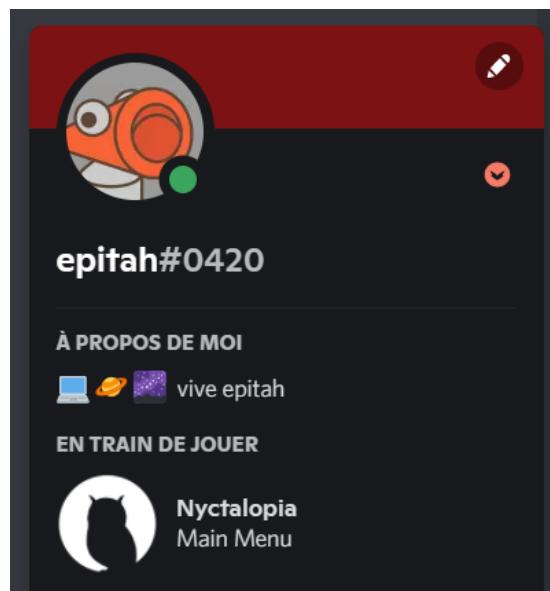


FIGURE 37 – *Discord Rich Presence*

### 3.8 Manuels d'instructions et d'installation

Nos manuels d'instructions et d'installation sont entièrement complets et vont permettre aux utilisateurs qui rencontreront des problèmes d'avoir tout de suite des réponses à leurs questions. On y retrouve une version en français mais aussi en anglais. Les utilisateurs pourront retrouver ces manuels sur la page de support du site web.



FIGURE 38 – Première page

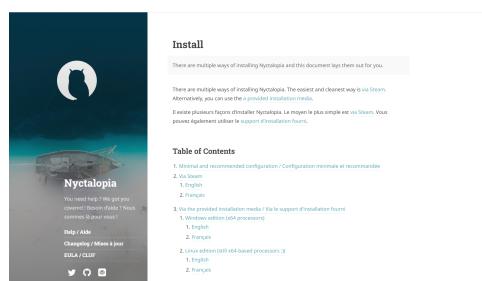


FIGURE 39 – Manuels d'instructions et d'installation sur le site web

### 3.9 Site Web

Notre site web est entièrement prêt au public. Nous avons également rajouté une page de support pour guider et aider les utilisateurs. Grâce à cette page ils pourront nous contacter, savoir comment marche le multijoueur, trouver des informations sur l'installation de Nyctalopia et retrouver la licence et condition d'utilisation du jeu. Cette fois le fond du site est agréable visuellement pour permettre aux utilisateurs de facilement lire les différentes informations présente sur cette page.

Nous avons essayé de rendre le site attractif. On peut le voir avec les effets qui permettent que l'arrière-plan et les images au premier plan ne défile pas à la même vitesse ou encore avec le curseur qui suit la souris. Pour ces effets, on a fait le choix d'utiliser des scripts JavaScript pour donner au site un aspect professionnel. De plus, le code HTML est très bien organisé pour nous permettre de l'améliorer ou de le modifier à tout moment. Nous avons choisi un thème sombre pour tout de suite mettre le joueur dans l'ambiance du jeu où il sera dans l'obscurité. Le site est complètement responsive, donc adapté à tout type d'écran pour permettre au joueurs de le consulter depuis n'importe où. A travers ce site, nous cherchons attirer les possibles joueurs en leur donnant envie de jouer à notre jeu, le tout en montrant notre professionnalisme.

En terme de contenu sur le site on peut retrouver :

- **Une page d'accueil** : Forêt animée



FIGURE 40 – Accueil du site

- **Une page de support** : On peut y accéder en appuyant sur **Support** dans la barre de navigation du site présente sur la page d'accueil.

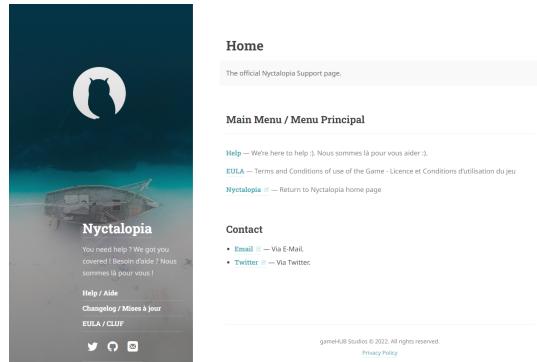


FIGURE 41 – *Page de support*

- **Images du jeu** : Des images de la forêt, l'interface utilisateur ou la scène d'introduction.

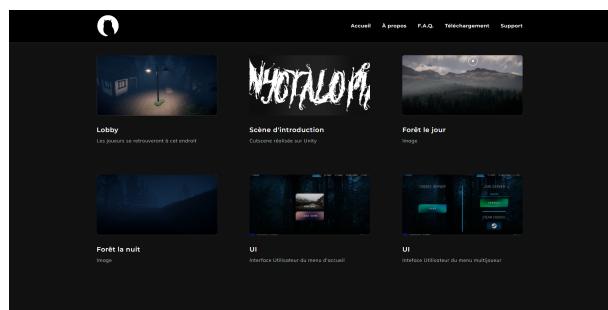


FIGURE 42 – *Images du jeu*

- **Contact** : Un formulaire dans lequel l'utilisateur écrit ses coordonnées et un bouton **Envoyer** qui renvoie le message écrit dans le cadre de texte vers notre boîte mail.

FIGURE 43 – *Contact*

- **Équipe** : Les rôles de chaque membre.



FIGURE 44 – *Membres du studio GameHub*

- **F.A.Q** : Les questions fréquemment posées.

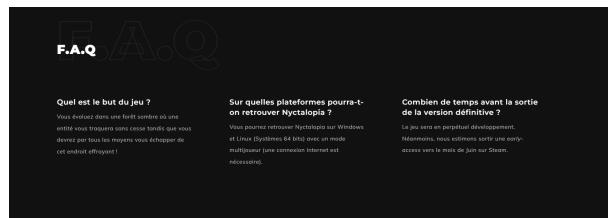


FIGURE 45 – *F.A.Q*

- **Téléchargements** : Lorsque l'utilisateur clique sur le lien **Télécharger Nyctalopia** il arrive sur cette page qui propose deux méthodes de téléchargement du jeu. Un lien pour le télécharger sur Steam et un lien Windows ou Linux suivant votre système d'exploitation.

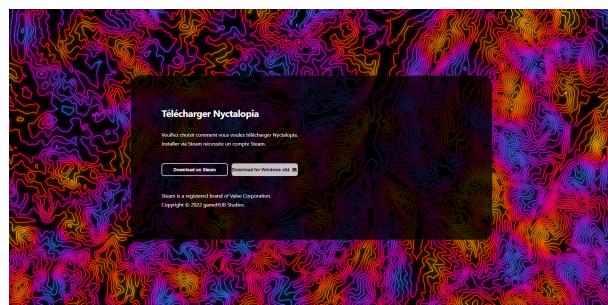


FIGURE 46 – *Téléchargements*

- **Annexe** : On y retrouve les liens vers la différente documentation du projet ainsi que des informations diverses.



FIGURE 47 – Annexe

Le site web est hébergé sur CloudFlare Pages, un service gratuit de l’entreprise de gestion DNS CloudFlare, permettant de déployer rapidement à partir d’un repo Git, un site web statique rapide grâce aux serveurs CDN disposés partout dans le monde et sécurisé par un certificat SSL et une protection anti-DDOS. Le nom de domaine choisi est *nyctalopia.games*. Il est simple et facile à retenir pour l’utilisateur.

Gestion DNS pour <b>nyctalopia.games</b>						
Type	Nom	Contenu	État du proxy	Durée TTL	Actions	
A	gitlab	[REDACTED]	DNS uniquement	Automatique	<a href="#">Modifier</a>	
CNAME	get	gitlab.nyctalopia.games	Proxied	Automatique	<a href="#">Modifier</a>	
CNAME	linux.get	gitlab.nyctalopia.games	Proxied	Automatique	<a href="#">Modifier</a>	
CNAME	nyctalopia.games	nyctalopia.pages.dev	Proxied	Automatique	<a href="#">Modifier</a>	

FIGURE 48 – Interface CloudFlare



FIGURE 49 – Site sécurisé SSL

Nous avons également décidé de mettre en place un GitLab (*gitlab.nyctalopia.games*) privé sur un serveur nous appartenant, ce qui nous a permis d’utiliser la puissance brute de Git LFS et des pipelines CI/CD, améliorant notre productivité sans perdre de temps.

## 4 Planning

Voici le planning d'avancement des tâches par période (temps séparant deux soutenances).

Tâches	Soutenance 1	Soutenance 2	Soutenance 3
AI - Intelligence Artificielle	10%	40%	80%
Audio	50%	80%	100%
Communication & Gameplay	30%	50%	80%
Graphismes & Modèles	50%	70%	100%
Manuels d'instruction	0%	100%	100%
Carte	15%	50%	90%
Multijoueur	80%	90%	90%
Sauvegardes	5%	40%	60%
UI/UX	80%	90%	60%
VFX	10%	50%	100%
Site web	90%	100%	100%

Code couleur :

- **Vert** : En avance
- **Rouge** : En retard

## 5 Difficultées

Lors du développement de ce jeu, nous avons rencontré divers problèmes qui nous ont déranger dans notre travail et réduit notre efficacité.

Le plus dérangeant de tout est le fait que nous avons décidé de faire un jeu en 3D avec des textures de bonne qualité. En effet, cela a rendu le jeu très lourd, au final nous avons réussi à réduire la taille du projet à environ 39 GB, mais lors du développement cela a empêché certaines personnes de pouvoir directement développer et tester sur le jeu causant des conflits entre les différentes implémentations. De plus, tout le groupe a passé un temps non-négligeable dans les écrans de chargement sur *Unity3D*, notamment lors de la compilation des scripts et du lancement des scènes. Des écrans de chargements dont la durée a été accentuée par le manque de puissance de nos ordinateurs, souvent portables.

De plus, concernant les sauvegardes, malgré le fait que les scripts et les classes soient finalisés, nous avons rencontré un problème concernant l'interface. En effet, lors de l'apparition de l'interface des sauvegardes, qui était censée afficher les trois emplacements de sauvegarde, nous n'avons pas réussit à garder cette interface au milieu de l'écran du joueur rendant son utilisation impossible car celle-ci était partiellement visible à l'écran. Nous n'avons, par conséquent, pas pu implémenter celle-ci dans cette version du jeu et donc malgré nos efforts, le système de sauvegardes n'est pas fonctionnel.

Beaucoup de changements nous ont provoqué des ralentissements tout au long du projet, notamment le multijoueur, dont le SDK a été modifié plusieurs fois pour avoir un multijoueur compatible avec notre jeu et avec Steam.

L'intelligence artificielle et l'algorithme de pathfinding nous ont aussi posés des problèmes qu'on a finalement su résoudre au détriment d'un système d'attaque, de mort et de réapparition fonctionnels. Notre créature suivra parfaitement le joueur dans le chapitre 3 en traçant le chemin le plus court pour l'atteindre, mais malheureusement celle-ci ne sera pas capable d'attaquer celui-ci.

Au commencement du projet, nous avions l'ambition d'implémenter quelques caractéristiques pouvant rendre l'expérience du joueur plus agréable tel que la personnalisation des touches. Malheureusement, nous nous sommes rendu compte, peut être trop tard, que cela n'était pas vital au fonctionnement du jeu et nous nous sommes plutôt concentrés sur les bases pour avoir un jeu fonctionnel.

## 6 Conclusion

Enfin, l'esprit de groupe a été également un point important de l'apprentissage c'est à dire savoir s'entraider, communiquer, s'organiser et planifier nos objectifs. Ce sont des points que l'on attend d'un ingénieur et qui nous seront demandés durant nos carrières professionnelles. Il est donc évident que la qualité de ce projet représente un reflet de nos capacités à évoluer au sein d'un groupe pour aller vers un même but.

Même si compliqué, ce projet nous a apporté beaucoup de connaissances, non seulement dans l'utilisation de logiciels 3D comme Unity ou dans C#, le langage de programmation associé à celui-ci. Ce projet nous a appris aussi à travailler en groupe et à s'organiser au sein de celui-ci pour obtenir des résultats satisfaisants.

## 7 Annexe

**Nyctalopia** : Traduction anglaise du mot héméralopie, qui est la cécité nocturne, ou l'incapacité à bien voir dans un éclairage sombre.

**Survival horror** : Le survival horror est un genre de jeu vidéo, sous-genre du jeu d'action-aventure, inspiré des fictions d'horreur. Bien que des aspects de combats puissent être présents dans ce type de jeu, le gameplay fait généralement en sorte que le joueur ne se sente pas aussi puissant qu'il ne le serait typiquement dans un jeu d'action, et ce en limitant par exemple la quantité de munitions, d'énergie ou de vitesse. Le joueur doit parfois chercher certains objets pour avoir accès à un passage vers une nouvelle zone, et résoudre des énigmes à certains moments. Les jeux utilisent des thèmes d'horreur, et le joueur est souvent confronté à des environnements obscurs et à des ennemis qui peuvent surgir de nulle part.

**Jump scare** : Un jump scare est un principe qui recourt à un changement brutal intégré dans une image, une vidéo ou une application pour effrayer brutalement le spectateur ou utilisateur. Ce principe s'est développé dans les années 1990 notamment au cinéma.

**UI - Interface Utilisateur** : L'interface utilisateur est un dispositif matériel ou logiciel qui permet à un usager d'interagir avec un produit informatique. C'est une interface informatique qui coordonne les interactions homme-machine, en permettant à l'usager humain de contrôler le produit et d'échanger des informations avec le produit.

**IP** : Une adresse IP est un numéro d'identification qui est attribué de façon permanente ou provisoire à chaque périphérique relié à un réseau informatique qui utilise l'Internet Protocol. L'adresse IP est à la base du système d'acheminement des paquets de données sur Internet. Il en existe deux versions : IPv4 et IPv6.

**DNS** : Le Domain Name System, généralement abrégé DNS, qu'on peut traduire en « système de noms de domaine », est le service informatique distribué utilisé pour traduire les noms de domaine Internet en adresse IP ou autres enregistrements.

Source : Wikipédia



Nyctalopia - gameHUB Studios

2022

EPITA Toulouse