Rapport 2 - Shattered Mind

DEVONKHAIN
Noémie DA CUNHA LOBO, Chloé BAUER, Séléna CONSTANS

15 janvier 2025

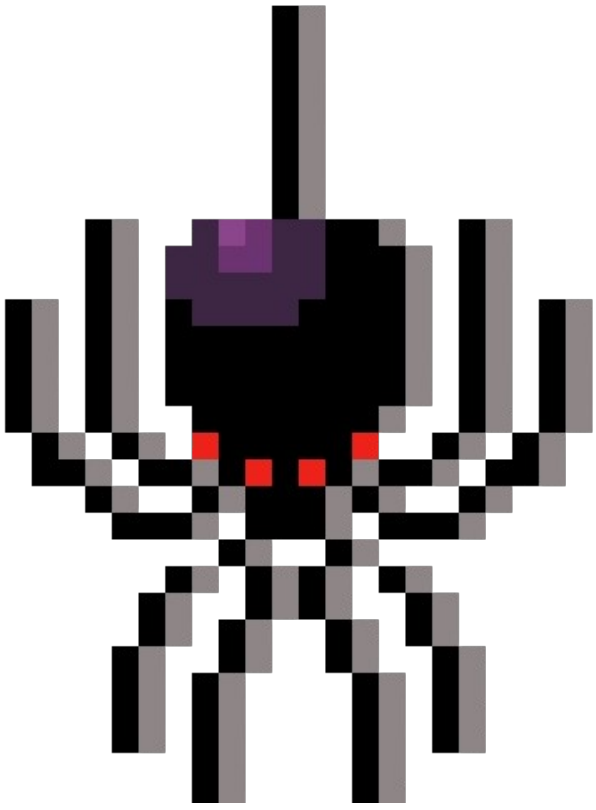


Table des matières

1 Introduction

Le jeu que nous proposons se nomme Shattered Mind, il sera un mélange d'horreur, d'énigmes et de combats avec une progression en étages dans lequel deux protagonistes doivent tout faire pour se retrouver et ainsi terminer le jeu. Nos parcours respectifs ont de nombreux points communs comme la passion du jeu, une soif de connaissance quant à la programmation et la création d'un jeu qui plaira au maximum de personnes. Cependant, nos parcours ont aussi chacun leurs propres différences ce qui ajoute grandement à la créativité au sein de notre entreprise et nos différences d'expérience nous permettent de trouver des solutions plus facilement pour permettre au projet de progresser au mieux.

2 Organisation et gestion du projet

2.1 Répartition des rôles

Noémie DA CUNHA LOBO occupe le rôle de cheffe d'équipe et supervise la section développement. Elle est responsable de la mise en œuvre des éléments fondamentaux du projet, tels que les fonctionnalités multijoueurs et l'intelligence artificielle (IA). De plus, elle veille à la maintenabilité du code ainsi qu'au respect des normes et conventions de développement.

Chloé BAUER est en charge de la direction artistique et du design, de plus elle fait également partie de l'équipe de développement et vérifie que tous les visuels soient biens implémentés dans le jeu. Elle conçoit la majorité des éléments visuels du jeu, s'assure de leur intégration harmonieuse dans l'environnement numérique, et garantit la cohérence des proportions entre les différents éléments visuels.
Tout design non accompagné par des crédits a été fait par Chloé BAUER.

Séléna CONSTANS, quant à elle, occupe le poste de responsable de la narration. Elle veille à ce que toutes les réalisations soient alignées avec l'histoire du jeu et que l'ensemble des éléments respecte une logique et une cohérence narrative.

2.2 Méthodologie de travail

Dans le cadre de ce projet, nous avons choisi de nous inspirer de la méthode Agile, et plus spécifiquement de celle de Kanban, qui correspondait le mieux aux habitudes et méthodes de travail de chacun.

Bien que nous ne suivions pas l'intégralité des recommandations proposées, nous avons fait le choix de conserver de nombreuses similitudes avec cette approche.

— Chaque membre de l'équipe dispose d'une liste de tâches à réaliser, sans être contraint par des périodes fixes comme dans un cadre de planification

en "sprints" (tel qu'en Scrum). Dès qu'une tâche est terminée, une autre peut être commencée. Toutefois, pour éviter toute surcharge de travail, les tâches sont priorisées. Nous essayons de ne pas entreprendre plus de 2 à 3 tâches simultanément, en fonction de la charge de travail qu'elles impliquent. Cette priorisation permet de maintenir un équilibre dans la répartition des tâches et d'éviter une trop grande accumulation de travail pour un seul membre à un moment donné.

— Nous avons mis en place un système de gestion des dépendances entre les tâches. Lorsqu'une tâche dépend de l'achèvement d'une autre, les membres concernés se coordonnent afin de garantir que les tâches soient terminées dans les délais et que l'avancement de chaque membre ne soit pas retardé. Cette gestion anticipée des dépendances assure une fluidité dans l'exécution du projet et permet de respecter les échéances.

— Nos réunions sont organisées de manière flexible, en fonction de l'avancement des travaux et de la charge de travail de chacun. À l'approche des soutenances, nous augmentons la fréquence des réunions pour nous préparer au mieux. Celles-ci ont pour but de définir les priorités, en tenant compte des tâches en cours et à venir, des tâches dépendantes et de l'avancement global du projet. Nous nous adaptons également à l'évolution du projet, ce qui signifie que des réunions peuvent être demandées à tout moment par n'importe quel membre du groupe si nécessaire.

— Afin de visualiser et organiser notre travail, nous avons recours à un tableau numérique via le logiciel Trello. Ce tableau nous permet de suivre l'état des tâches, en les classant en trois catégories principales : tâches accomplies, tâches en cours et tâches à faire. Grâce à cette visualisation claire, nous pouvons également hiérarchiser les tâches, ce qui nous aide à concentrer nos efforts sur les éléments les plus importants du projet à chaque étape.

Dans cette section, nous présenterons les parties du projet qui sont achevées ou presque terminées. Les explications techniques détaillées de ces différentes composantes seront disponibles dans la section "3.3 Bilan technique".
Voici les tâches menées à bien jusqu'à présent :

— **Noémie Da Cunha Lobo**
Elle a pris en charge l'implémentation des fonctionnalités multijoueurs ainsi que de l'intelligence artificielle, garantissant ainsi une interaction fluide entre les joueurs et une réactivité optimale du système.

— **Chloé BAUER**
Elle était responsable de l'ensemble des éléments graphiques du projet et veillait à leur cohérence avec le jeu. Son travail consistait notamment à établir un lien entre les visuels et le gameplay afin d'assurer une inté-

gration harmonieuse. Cette mission de coordination a été réalisée en collaboration avec Noémie Da-Cunha-Lobo et Séléna Constans, qui étaient en charge de l'utilisation d'Unity ainsi que de la résolution des éventuels problèmes techniques liés au logiciel et à l'affichage des visuels.

— **Séléna Constans**
Elle s'est occupée du codage de différentes énigmes que doivent résoudre Léon et Yorick mettant en place des mécanismes variés.

3 Avancements réalisés depuis la dernière soutenance

3.1 Résultats intermédiaires

— **Multijoueur**

Nous avons opté pour un mode multijoueur Peer-to-Peer (P2P), où l'hôte de la partie joue également le rôle d'hébergeur. Dans cette configuration, l'hôte est responsable de l'hébergement de la session de jeu, ce qui signifie qu'il gère les connexions et synchronise les données entre tous les joueurs participants. Les autres joueurs se connectent directement à cet hôte pour échanger les informations nécessaires au bon déroulement du jeu. Cette approche permet de réduire les coûts liés à l'hébergement tout en garantissant une interaction fluide entre les joueurs.

Le multijoueur devant être ajusté progressivement, cette étape constitue une première configuration visant à intégrer tous les éléments nécessaires à son bon fonctionnement.

Nous utilisons "Netcode for GameObjects" pour la gestion du multijoueur. Afin d'assurer son bon fonctionnement, l'importation de plusieurs packages spécifiques est nécessaire dans notre projet. Ces packages permettent d'établir une communication efficace entre les joueurs et de synchroniser les éléments du jeu en réseau.

Pour effectuer cette étape, plusieurs solutions s'offrent à nous. Il est possible de télécharger manuellement chaque package un par un, ce qui permet un contrôle précis des éléments ajoutés. Une alternative plus efficace consiste à utiliser le "Multiplayer Center", qui analyse les besoins du projet en fonction du type de jeu développé et propose automatiquement les packages requis avant de procéder à leur importation. Cette seconde approche facilite l'intégration et réduit les risques d'omission de dépendances essentielles.

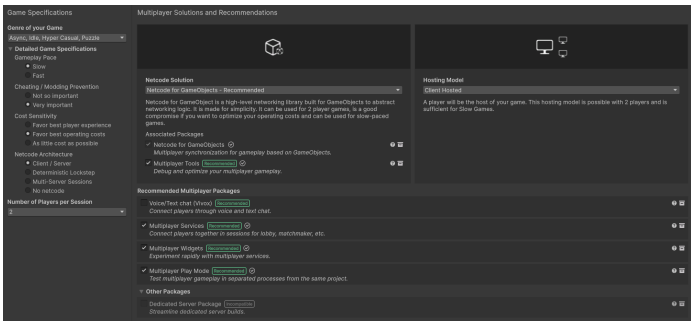


FIGURE 1 – Multiplayer Center

Au terme de cette configuration, cinq packages doivent être intégrés au projet afin d'assurer une gestion optimale du multijoueur :

— **Netcode for GameObjects :** pour la mise en réseau et la synchronisation des objets.
— **Multiplayer Services :** pour la gestion des services en ligne liés au multijoueur.
— **Multiplayer Widgets :** fournissant des outils d'interface dédiés au développement multijoueur.
— **Multiplayer Play Mode :** permettant de tester le mode multijoueur en local.
— **Multiplayer Tools :** regroupant divers outils d'analyse et de débogage pour le multijoueur.

Ces composants sont essentiels pour assurer une expérience fluide et fonctionnelle en mode multijoueur.

Il est également indispensable d'importer le package "TextMesh Pro", qui est requis pour certains composants du projet. Ce package est particulièrement utile pour la gestion et l'affichage de textes enrichis, notamment dans le cadre de la création de sessions multijoueurs, où une interface claire et lisible est essentielle.

Pour tester le jeu avec deux joueurs sur deux écrans distincts, il est nécessaire d'utiliser la fenêtre "Multiplayer Play Mode". Pour y accéder, il faut suivre le chemin suivant : Window → Multiplayer → Multiplayer Play Mode

Dans cette fenêtre, il suffit ensuite de cocher l'option "Player 2". Cela permet d'ouvrir une deuxième instance d'Unity, rendant possible le test du jeu en multijoueur. Une instance peut alors être configurée en tant qu'hôte, tandis que l'autre fonctionnera en tant que client, permettant ainsi de vérifier la bonne communication entre les deux sessions.

— **Menu principal**

— **Network Manager**

L'intégralité du système multijoueur du jeu est gérée depuis le menu principal. En effet, des objets essentiels, tels que le NetworkManager, sont configurés directement dans la scène du menu principal, qui est la première scène à être chargée. Ces objets sont ensuite transférés d'une scène à l'autre tout au long du jeu, assurant ainsi la continuité et la stabilité de l'expérience multijoueur, quel que soit l'endroit où le joueur se trouve dans le jeu.

Dans un premier temps, il est nécessaire de créer un objet vide dans Unity en suivant la procédure suivante : GameObject → Create Empty. Une fois l'objet créé, il convient de le nommer NetworkManager. Ensuite, il faut lui associer deux scripts essentiels : le script "NetworkManager", fourni avec les packages importés précédemment, pour gérer l'ensemble des fonctionnalités liées au réseau (connexion des joueurs, synchronisation des objets, gestion des scènes en multijoueur), et le script "Unity Transport", qui permet de gérer la communication réseau en utilisant le protocole de transport d'Unity, assurant ainsi une transmission stable et rapide des données entre les clients et l'hôte du jeu.

Il n'est pas nécessaire de modifier les réglages par défaut du script "Unity Transport". En ce qui concerne le NetworkManager, il est essentiel de s'assurer que la topologie du réseau soit définie sur "Client-Server" dans les paramètres de Network Topology. De plus, il n'est pas nécessaire de définir un Player Prefab par défaut, car deux préfabs ont été créés, un pour chaque joueur. Ces préfabs seront assignés dynamiquement lors de la connexion des joueurs, garantissant ainsi une gestion correcte des différentes instances de joueur dans le jeu. Cependant, il est nécessaire de placer les deux préfabs dans la Network Prefabs List. Cela permet au NetworkManager de reconnaître et de gérer correctement les instances des joueurs lorsqu'ils rejoignent ou hébergent une partie. En ajoutant ces préfabs à cette liste, le système peut automatiquement synchroniser les objets des joueurs à travers le réseau, garantissant ainsi leur apparition dans les scènes appropriées et leur gestion correcte en mode multijoueur.

— **Game Manager**

Le GameManager est responsable de l'attribution des préfabs aux joueurs et du spawn des joueurs dans la scène suivante.

Comme pour le NetworkManager, il faut créer un objet vide dans Unity en suivant la procédure suivante : GameObject → Create Empty. Cet objet doit être nommé GameManager. Ensuite, il est nécessaire de lui associer le script SC_GameManager, ainsi que le composant "Network Object". Ce dernier est indispensable pour que l'objet GameManager soit reconnu et synchronisé sur le réseau, permettant ainsi

une gestion correcte du spawn des joueurs dans les différentes scènes du jeu.

```
public class SC_GameManager : NetworkBehaviour
{
    1 référence
    [SerializeField] private Transform Player1;
    1 référence
    [SerializeField] private Transform Player2;

    0 références
    public override void OnNetworkSpawn()
    {
        if (IsServer)
            NetworkManager.Singleton.SceneManager.OnLoadEventCompleted += SceneLoaded;
    }

    1 référence
    private void SceneLoaded(string sceneName, LoadSceneMode loadSceneMode, List<ulong> clientsCompleted, List<ulong> clientsTimeOut)
    {
        foreach (ulong clientId in NetworkManager.Singleton.ConnectedClientsIds)
        {
            Transform playerTransform = clientId == 0 ? Instantiate(Player1) : Instantiate(Player2);
            playerTransform.GetComponent<NetworkObject>().SpawnAsPlayerObject(clientId, true);
        }
    }
}
```

FIGURE 2 – Script GameManager

— **Gestion de la session**

Pour la création et la gestion des sessions multijoueurs, nous avons décidé que l'ensemble du processus se déroulerait dans le menu principal. Ainsi, le bouton "Jouer" ne mène plus directement à la scène suivante, mais fait apparaître deux nouveaux boutons. Ces boutons permettent aux joueurs de choisir s'ils souhaitent héberger une partie en tant qu'hôte ou rejoindre une partie existante. Cette approche centralise la gestion des sessions et offre une meilleure expérience utilisateur en permettant aux joueurs de choisir facilement leur rôle dans la partie multijoueur avant de commencer.

Le package "Multiplayer Widgets" facilite l'importation directe des widgets nécessaires pour la gestion du multijoueur. Il permet notamment de créer le code permettant aux joueurs de rejoindre une partie, d'afficher un encadré où les joueurs peuvent entrer un code de session pour se connecter, ainsi que d'afficher une liste des joueurs actuellement présents dans la session. Ces widgets sont essentiels pour simplifier l'interface utilisateur et offrir une expérience de jeu multijoueur plus fluide et intuitive.

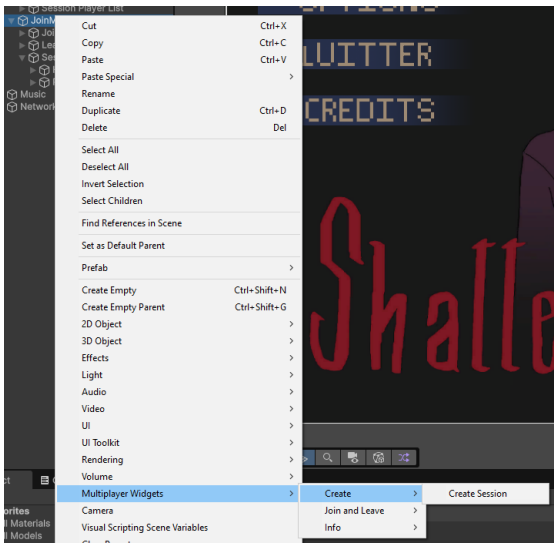


FIGURE 3 – Multiplayer Widgets

FIGURE 4 – Hôte

FIGURE 5 – Client

— **Personnages**

Pour chaque personnage, il est nécessaire d'ajouter les composants suivants : "Network Object", "Network Transform", et "Network Rigidbody 2D". Ces composants sont essentiels pour garantir la synchronisation correcte des objets à travers le réseau.

— **Network Object** permet de gérer l'état du joueur sur le réseau, garantissant que l'objet est bien synchronisé et reconnu par les autres joueurs.

— **Network Transform** permet de synchroniser la position, la rotation et l'échelle du personnage en temps réel entre les différentes instances du jeu.

— **Network Rigidbody 2D** est utilisé pour gérer la physique du personnage en 2D et s'assurer que les forces physiques appliquées sont correctement synchronisées.

Il est également crucial de vérifier que, dans le Network Object, le hash de chaque personnage soit unique. Cela garantit qu'il n'y a pas de conflits d'identification entre les instances de personnages, assurant ainsi une gestion précise des objets multijoueurs.

Pour le Network Transform, il est important de configurer le Authority Mode sur Owner, ce qui signifie que l'autorité sur la position de l'objet (dans ce cas, le personnage) sera donnée au joueur qui en est le propriétaire. Cela permet au joueur de contrôler directement la position de son personnage, tandis que les autres joueurs synchroniseront cette position à travers le réseau.

De plus, pour optimiser les performances et éviter une synchronisation inutile, seules les positions x et y doivent être synchronisées. Cela signifie que la position en z (pour les jeux en 2D) ou la rotation et l'échelle ne seront pas prises en compte pour la synchronisation réseau, réduisant ainsi la charge de synchronisation et assurant une expérience plus fluide

en multijoueur.

En ajoutant le script SC_OwnerNetworkAnimator à chaque personnage, celui-ci permet de gérer l'animation des personnages de manière réseau tout en utilisant le système de NetworkAnimator d'Unity. Ce script, qui hérite de NetworkAnimator, redéfinit la méthode OnIsServerAuthoritative(), en retournant false. Cela signifie que l'animation du personnage est contrôlée par le client, et non par le serveur, garantissant ainsi que l'hôte n'a pas l'autorité sur les animations des autres joueurs.

En d'autres termes, avec ce script, chaque client sera responsable de l'animation de son propre personnage, tandis que le serveur coordonne les autres aspects du gameplay. Cela permet une gestion plus fluide des animations en multijoueur, tout en évitant les conflits de synchronisation.

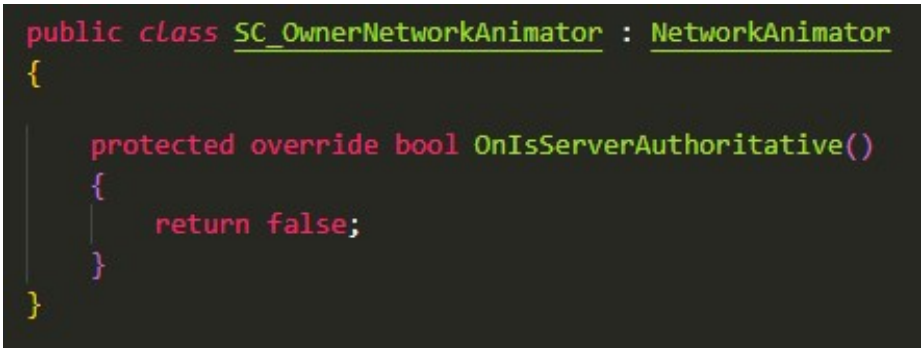


FIGURE 6 – Script OwnerNetworkAnimator

Dans les propriétés de l'ennemi, il est nécessaire d'ajouter les composants suivants :

— **Box Collider 2D** : Ce composant permet d'affecter une boîte de collision à l'ennemi, ce qui lui permet d'interagir correctement avec le monde du jeu et les autres objets (comme les obstacles ou les projectiles). La Box Collider 2D définit une zone de détection autour de l'ennemi pour gérer les collisions.

— **Rigidbody 2D** : Ce composant permet à l'ennemi de bénéficier des lois de la physique dans un environnement 2D. Le Rigidbody 2D permet à l'ennemi de se déplacer en fonction des forces physiques appliquées, comme la gravité ou d'autres forces. Il est également utilisé pour gérer la vitesse et la direction de l'ennemi lors de ses déplacements, ce qui est essentiel pour que les mouvements soient réalistes et cohérents avec l'algorithme de pathfinding.

Ces deux composants sont cruciaux pour assurer un comportement correct et une interaction avec l'environnement, notamment en matière de collision et de mouvement dans un jeu en 2D.

FIGURE 7 – menu

Nous avons effectué des changements pour le menu. Pour suivre mieux la Direction Artistique, nous avons changé la couleur des boutons. Ils passent donc d'un bleu foncé à un rouge, de cette manière ils correspondent mieux à l'image de Shattered Mind. De plus, des ajouts de détails comme des ombres sur le pull ont pu être faite afin que l'image aie un rendu moins plat. Enfin l'image d'origine à été assombrie pour que le nom du jeu ressorte mieux et qu'on ressente mieux le côté sombre et effrayant du jeu.

FIGURE 8 – menu

Ensuite après le lancement dans le menu on tombe sur la 1ere cinématique du jeu : il s'agit des 2 personnages principaux qui discutent pour savoir comment entrer dans le manoir. L'idée ici est de toujours garder un aspect sombre et la colorisation des canapée est due encore une fois à la Direction Artistique. Le choix de mettre peu de détail à été fait pour se focaliser sur les 2 protagonistes et donc mettre en lumière leur discussion.

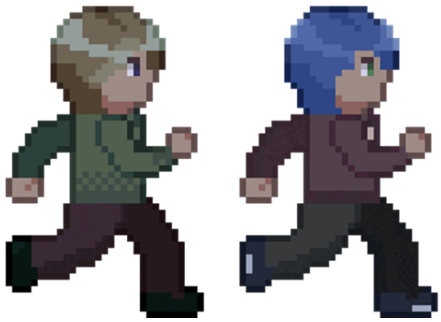


FIGURE 9 – chargement

Pour l'image de chargement entre le menu et la première scène, nous avons décidé de mettre les 2 personnages l'un à côté de l'autre en courant. Nous l'avons créée à partir des animations de marches des protagonistes. Ce choix s'interprète de la sorte qu'il s'agisse d'un jeu coopératif et que chacun des 2 joueurs est important et que, « main dans le main » ils arriveront à finir le jeu ensemble.

FIGURE 10 – grenier

Une grande partie de la map à été créée, donc ici nous avons le grenier, c'est à cet endroit que les joueurs apparaitront dans la 1ère scène. On aperçoit que la 1ère scène qui accueille les joueurs et plutôt encombrée et met en place un grenier abandonné avec des affaires d'une époque ancienne. On veut faire comprendre aux joueurs que cette pièce est un endroit remplit d'affaires peu importantes et dont on a voulu oublier en les entreposant dans le grenier du manoir.

Ensuite, nous avons aussi le 1er étage, constitué du couloir, de la salle de bain, la chambre parentale et la chambre d'enfant.

C'est donc le couloir qui permet d'accéder :

— Escalier de gauche : le grenier
— Escalier de droite : le rez-de-chaussée
— Porte gauche : la chambre pour enfant
— Porte droite : la salle de bain
— Porte en bas : la chambre parentale

FIGURE 11 – couloir

Voici les pièces citées :

FIGURE 12 – salle de bain

FIGURE 13 – chambre parentale

Les animations ont vu elles aussi une évolution et ont été complètement changées. La principale évolution vient du fait que l'ancienne animation était sur 4 frames, tandis que la nouvelle version est sur 6 frames, ce qui rend le mouvement plus fluide et agréable à regarder. L'ennemi présenté lors de la dernière soutenance à lui aussi eu une animation pour les déplacements qu'il aura dans le jeu.

FIGURE 14 – chambre d'enfant

FIGURE 15 – animations Yorick

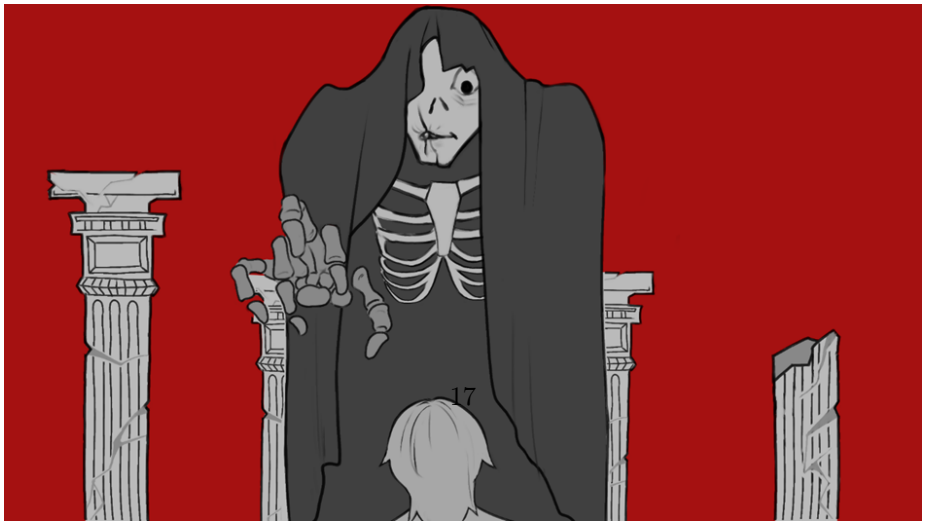


FIGURE 18 – cinématique boss

FIGURE 16 – animations Léon

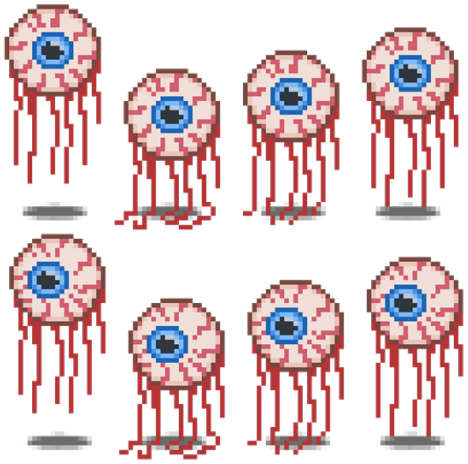


FIGURE 17 – Animation ennemi

Enfin nous avons tout autant avancé sur la fin du jeu avec la cinématique du boss final. Celle-ci n'est pas aboutie mais elle en demeure entamée. Au centre de l'image il s'agit donc du boss, et en bas au milieu de Yorick face à la créature.

— **Intelligence artificielle**

Pour gérer l'intelligence artificielle (IA) dans le jeu, nous utilisons le système "A* Pathfinding". Dans la scène de jeu, il est nécessaire de créer un objet vide que nous nommerons A*. À cet objet, il faut ensuite associer le composant Pathfinder. Ce composant permet de gérer les déplacements des entités contrôlées par l'IA en utilisant l'algorithme A*, qui calcule le chemin optimal pour atteindre un objectif en tenant compte des obstacles et du terrain. Cette approche garantit des déplacements fluides et intelligents pour les personnages non-joueurs (PNJ) dans le jeu. Une fois le composant Pathfinder associé à l'objet A*, il faut ouvrir les paramètres du composant et cliquer sur l'option "Graphs". Ensuite, sélectionnez "Grid Graph". Cela permettra de définir la grille de navigation utilisée par l'algorithme A* pour calculer les chemins. Le Grid Graph divise la scène en une grille de cellules, où chaque cellule représente une unité de terrain que l'IA peut parcourir. Cela permet de créer une carte de navigation efficace et de gérer les déplacements de l'IA en fonction des obstacles présents dans l'environnement du jeu.

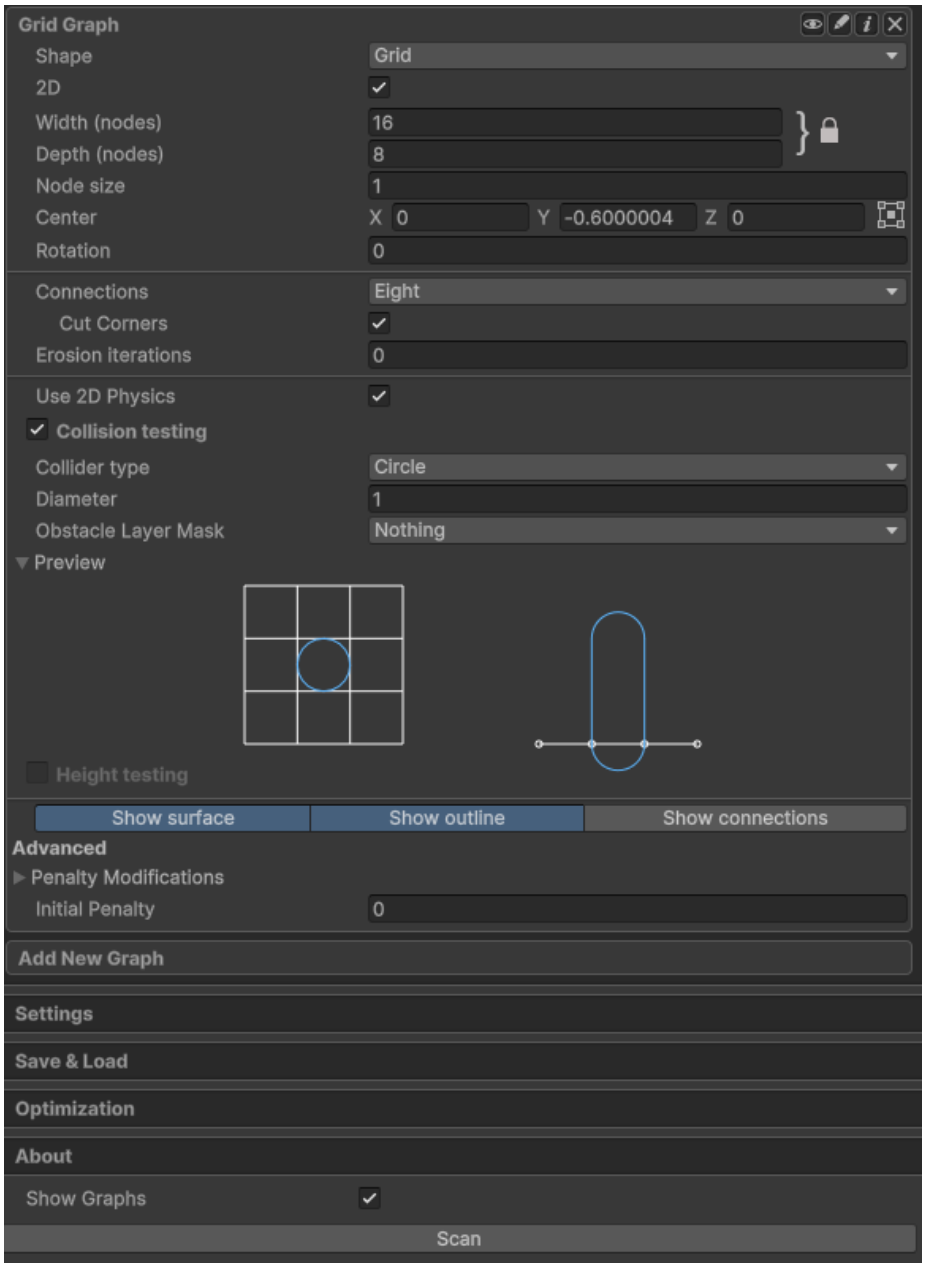


FIGURE 19 – Pathfinder

Une fois que le Grid Graph est sélectionné dans le composant Pathfinder, il est nécessaire de modifier les paramètres "Width", "Depth" et "Center" pour ajuster la zone d'action de l'IA :

— **Width** définit la largeur de la grille en nombre de cellules. Cela détermine combien d'unités seront présentes horizontalement dans la zone de navigation.
— **Depth** définit la profondeur de la grille en nombre de cellules. Cela détermine combien d'unités seront présentes verticalement dans la zone de navigation.
— **Center** permet de définir le point central de la grille, ce qui permet d'ajuster l'emplacement de la zone de navigation dans la scène.

Ces ajustements permettent de définir précisément la zone où l'IA pourra se déplacer, en prenant en compte les dimensions et l'emplacement du terrain ou de la carte de jeu. Ces paramètres doivent être configurés en fonction de la taille et des besoins spécifiques de la scène de jeu. Une fois cela fait, il faut appuyer sur "Scan".

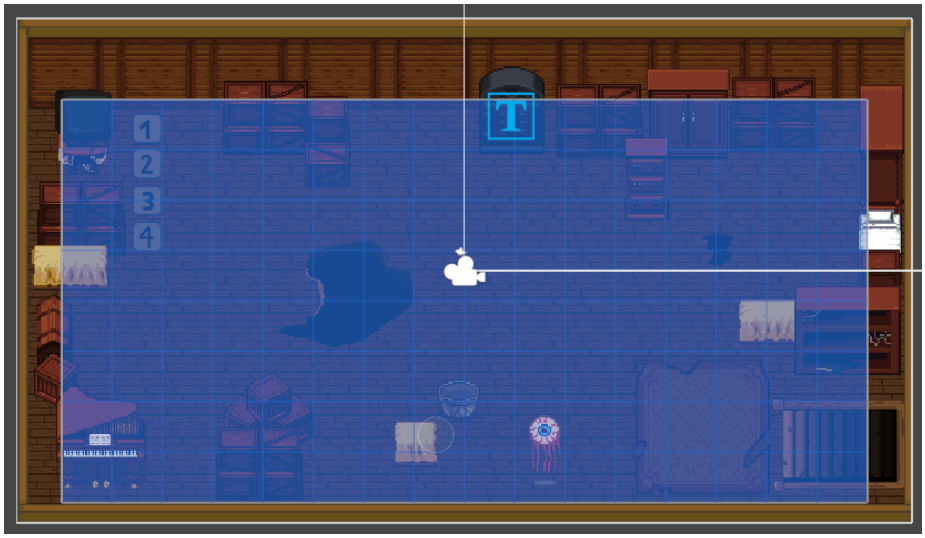


FIGURE 20 – Zone d'action

Dans les propriétés de l'ennemi, il est nécessaire d'ajouter les scripts "Seeker" et "EnemyAI" pour gérer le comportement de l'IA.

— **Seeker** : Ce composant est utilisé pour calculer les chemins entre l'ennemi et sa cible (par exemple, le joueur). Il permet de déterminer le meilleur chemin à suivre en fonction des obstacles et du terrain, et ce chemin est mis à jour à intervalles réguliers (toutes les 0,5 secondes dans ce cas).
— **EnemyAI** : Ce script gère le comportement spécifique de l'ennemi. Il permet à l'ennemi de se déplacer vers la cible en suivant un chemin calculé par le Seeker. Il contrôle également la vitesse de déplacement de l'ennemi, la distance à laquelle il change de waypoint, et la distance minimale pour déclencher une attaque (si l'ennemi est assez proche de

la cible). Si le joueur est hors de portée d'attaque, l'ennemi poursuit la cible jusqu'à ce qu'il soit suffisamment proche pour effectuer une action d'attaque.

Ce système permet à l'ennemi de naviguer de manière intelligente en utilisant l'algorithme A*, tout en réagissant aux mouvements du joueur et en s'arrêtant pour attaquer lorsque nécessaire.

Dans le script EnemyAI, il est en effet nécessaire d'avoir un seul target, car à l'avenir, les personnages seront répartis dans des pièces différentes et ne se rencontreront pas. Par conséquent, les ennemis n'auront pas besoin de suivre plusieurs cibles simultanément. Cela simplifie la gestion de l'IA, car chaque ennemi sera affecté à un seul target spécifique à la scène dans laquelle il se trouve.

```
public class EnemyAI : MonoBehaviour
{
    // Cible que l'ennemi doit suivre (par exemple, le joueur)
    2 références
    public Transform target;

    // Vitesse de déplacement de l'ennemi
    1 référence
    public float speed = 120f;

    // Distance à laquelle l'ennemi considère qu'il a atteint un waypoint
    1 référence
    public float nextWpDistance = 1f;

    // Distance minimale pour déclencher une attaque (l'ennemi s'arrête en dehors de cette portée)
    1 référence
    public float attackRange = 2f;

    // Chemin calculé par le Seeker
    5 références
    public Path path;

    // Indice du waypoint actuel que l'ennemi essaie d'atteindre
    5 références
    int currWp = 0;

    // Composant Seeker utilisé pour calculer le chemin
    2 références
    public Seeker seeker;

    // Composant Rigidbody2D utilisé pour le mouvement physique de l'ennemi
    5 références
    public Rigidbody2D rb;

    // Méthode appelée au début de l'exécution
    0 références
    void Start()
    {
        // Met à jour le chemin toutes les 0,5 secondes pour suivre la position du joueur
        InvokeRepeating("UpdatePath", 0, 0.5f);
    }

    // Méthode pour mettre à jour le chemin vers la cible
    0 références
    void UpdatePath()
    {
        // Vérifie si le Seeker est prêt à calculer un nouveau chemin
        if (seeker.IsDone())
            // Demande un nouveau chemin du Seeker entre la position actuelle et la cible
            seeker.StartPath(rb.position, target.position, OnPathComplete);
    }
```

FIGURE 21 – Script EnnemyAI - 1

```
    // Méthode appelée lorsque le Seeker a terminé de calculer un chemin
    1 référence
    void OnPathComplete(Path p)
    {
        // Si le calcul a réussi (pas d'erreur), met à jour le chemin et réinitialise l'indice du waypoint
        if (!p.error)
        {
            path = p;
            currWp = 0;
        }
    }

    // Méthode appelée à chaque frame fixe pour gérer les mouvements physiques
    0 références
    void FixedUpdate()
    {
        // Si aucun chemin n'a été calculé ou si tous les waypoints ont été atteints, ne fait rien
        if (path == null || currWp >= path.vectorPath.Count)
        {
            return;
        }

        // Calcule la distance entre l'ennemi et le joueur
        float playerDistance = Vector2.Distance(target.transform.position, transform.position);

        // Si le joueur est en dehors de la portée d'attaque = on le poursuit
        if (playerDistance > attackRange)
        {
            // Calcule la direction vers le prochain waypoint
            Vector2 direction = ((Vector2)path.vectorPath[currWp] - rb.position).normalized;

            // Lisser la direction pour éviter des changements brusques (interpolation)
            Vector2 smoothDirection = Vector2.Lerp(rb.linearVelocity.normalized, direction, 0.1f);

            // Calcule la vitesse en fonction de la direction et de la vitesse spécifiée
            Vector2 velocity = smoothDirection * speed * Time.fixedDeltaTime;

            // Applique la vitesse calculée au Rigidbody2D
            rb.linearVelocity = velocity;

            // Calcule la distance entre l'ennemi et le waypoint actuel
            float distance = Vector2.Distance(rb.position, path.vectorPath[currWp]);

            // Si l'ennemi est suffisamment proche du waypoint, passe au suivant
            if (distance < nextWpDistance)
            {
                currWp++;
            }
        }
    }
}
```

FIGURE 22 – Script EnnemyAI - 2

— **Enigmes**

— Code Crypté
l'énigme du codé crypté comprend l'implémentation d'un Codelock.Pour implémenter un CodeLock dans Unity, il est nécessaire de mettre en place un système qui permet au joueur de saisir un code numérique spécifique afin de déverrouiller un objet ou d'activer une certaine fonction. Tout d'abord, il faut concevoir un ensemble de touches qui constitue un clavier numérique et une zone pour un afficheur où le code saisi sera affiché. Cela peut être fait avec Canvas, des boutons pour les chiffres, et un composant TextMeshPro pour la représentation physique correspondant aux boutons cliqués par le joueur.

Ensuite, un script est nécessaire pour gérer la logique du verrou. Ce script stocke une chaîne secrète et compare chaque entrée du joueur à cette chaîne. Lorsque le joueur clique sur l'un des boutons, le chiffre correspondant est ajouté à l'affichage. Ensuite, un bouton de validation devra être utilisé pour comparer le code saisi avec le mot de passe. S'ils correspondent, une action peut se produire, comme ouvrir une porte, afficher un message ou activer un mécanisme. Si le code ne correspond pas, le joueur recevra une notification d'erreur et peut choisir d'effacer ce qu'il a entré pour recommencer afin de trouver la bonne combinaison.

L'énigme comprend également du côté de l'autre joueur, les chiffres correspondant au mot de passe du CodeLock. Pour implémenter les chiffres correspondant au code que Yorick doit obtenir, il est nécessaire de créer un script permettant de faire clignoter et disparaître ces chiffres à tour de rôle. Le clignotement est réalisé en alternant l'affichage des chiffres à intervalles réguliers. Pour cela, un script attaché aux chiffres gère leur visibilité en activant et désactivant leur rendu à une fréquence déterminée. Cette alternance crée un effet dynamique pour attirer l'attention du joueur.

— Salle de Bain
L'énigme de la salle de bain repose sur un curseur de température que le joueur doit ajuster pour atteindre la valeur correcte. Au début, le joueur voit un thermomètre numérique affichant la température actuelle de l'eau et peut l'augmenter ou la diminuer à l'aide d'un slider. L'objectif est de régler la température sur 42°C par exemple, qui est la valeur correcte pour faire apparaître un indice caché.

Cependant, il ne suffit pas d'atteindre la température exacte : le joueur doit maintenir cette valeur pendant 2 secondes consécutives pour valider l'énigme. Si la température est modifiée avant la fin du timer, le compte à rebours se réinitialise et le joueur doit recommencer. Une fois

le temps écoulé avec la bonne température, un texte d'indice s'affiche à l'écran, indiquant un élément crucial pour progresser dans l'aventure. Cette mécanique oblige le joueur à faire preuve de précision et de patience pour découvrir l'information nécessaire à la suite du jeu.

— Radio
L'énigme de la radio est un puzzle basé sur la précision et la patience où le joueur doit régler trois fréquences radio correctes à l'aide de sliders. Au début, seul le premier slider de fréquence est visible, affichant une plage de valeurs allant de 30 MHz à 110 MHz. Le joueur doit ajuster ce slider pour atteindre la fréquence exacte, par exemple 97 MHz, et maintenir cette valeur pendant 3 secondes pour valider son choix. S'il bouge le slider avant la fin du timer, le compte à rebours est réinitialisé.

Une fois la première fréquence validée, le deuxième slider apparaît, et le processus recommence avec une nouvelle fréquence à trouver . Enfin, après avoir correctement réglé la troisième fréquence et maintenu la valeur pendant le temps imparti, un message de succès s'affiche, indiquant que toutes les fréquences ont été captées avec succès. Cette énigme repose sur la précision dans le réglage, la stabilité dans l'ajustement des valeurs, et l'observation pour trouver les bonnes fréquences, rendant la mécanique immersive et engageante pour le joueur.

— Cassette
L'énigme des cassettes repose sur un système d'indices interactifs où le joueur doit insérer différentes cassettes audio pour révéler des messages cachés. Au début, le joueur voit un lecteur de cassettes avec plusieurs boutons correspondant à différentes cassettes numérotées avec le chiffre attribué aux cassettes. Lorsqu'il appuie sur l'un des boutons, un texte d'indice apparaît à l'écran.

Chaque cassette contient un indice unique, et le joueur doit examiner attentivement ces informations pour avancer dans l'aventure. Cette énigme encourage l'exploration et la réflexion en obligeant le joueur à tester plusieurs cassettes et croiser les informations pour résoudre une énigme plus large.

— Horloge
L'énigme des horloges est un puzzle basé sur la coordination entre trois horloges interactives, réparties dans trois scènes différentes, et une horloge automatique, HorlogeBroken située dans une scène séparée.

Le joueur doit trouver et régler les bonnes heures sur les trois horloges interactives, chacune étant bloquée tant qu'elle n'est pas correctement réglée. Lorsqu'une horloge est réglée sur la bonne heure, elle se verrouille et ne peut plus être modifiée, déclenchant ainsi le changement vers la scène

suivante où se trouve l'horloge suivante.

Pendant ce temps, HorlogeBroken tourne en continu mais s'arrête automatiquement aux heures correspondant aux trois horloges interactives. Elle ne peut pas être contrôlée directement par le joueur et sa progression dépend entièrement de la validation des trois horloges. Lorsqu'une horloge manuelle est réglée correctement dans sa scène respective, HorlogeBroken reprend sa rotation et avance jusqu'à atteindre la prochaine heure clé, où elle s'arrête à nouveau en attente de la validation de l'horloge suivante.

Une fois que les trois horloges manuelles ont été réglées et que HorlogeBroken a marqué un arrêt aux trois heures correspondantes, elle effectue un dernier mouvement pour atteindre une heure finale, déclenchant ainsi l'ouverture d'une porte et permettant au joueur de progresser.

3.2 Bilan technique

Le projet a connu une avancée significative sur plusieurs aspects techniques, notamment en matière de multijoueur, d'intelligence artificielle et d'optimisation globale.

L'intégration du mode multijoueur repose sur un système Peer-to-Peer (P2P), utilisant Netcode for GameObjects pour la gestion des connexions et synchronisations. Un Network Manager a été mis en place pour organiser les sessions de jeu, permettant aux joueurs d'héberger ou de rejoindre une partie directement depuis le menu principal. La synchronisation des personnages est fluide, assurant des déplacements et interactions cohérents en réseau.

Du côté du gameplay, les personnages et leurs animations ont été synchronisés en multijoueur, garantissant une cohérence visuelle et fonctionnelle. L'intelligence artificielle des ennemis a été optimisée grâce à l'algorithme A*, leur permettant de détecter et poursuivre les joueurs avec précision.

Les énigmes et interactions ont été enrichies pour offrir une expérience de jeu plus engageante. Plusieurs énigmes nécessitent des actions spécifiques pour être résolues, qu'il s'agisse de manipuler des codes, d'ajuster des paramètres ou d'interagir avec l'environnement. Une énigme en particulier exige que le joueur maintienne une température donnée pour progresser.

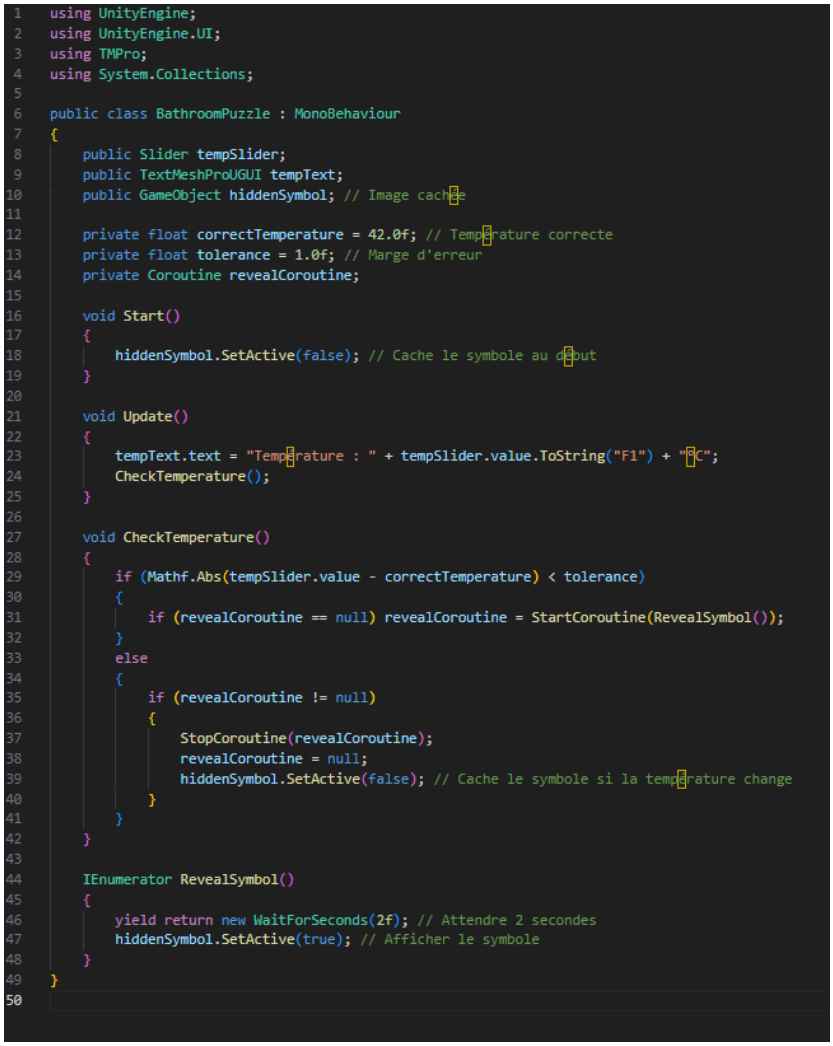
L'interface utilisateur a également bénéficié d'une refonte. Les menus ont été retravaillés pour une meilleure ergonomie, avec des ajustements de couleurs et de mise en page afin d'améliorer la lisibilité. L'intégration de cinématiques renforce l'immersion, rendant la progression plus fluide et narrative.

Enfin, des optimisations techniques ont été réalisées pour améliorer la fluidité et la stabilité du jeu. La gestion du réseau a été affinée pour éviter les problèmes de latence et garantir une expérience en ligne plus stable.

Ces avancées permettent au projet de se rapprocher d'une version plus aboutie, tant sur le plan technique que sur celui de l'expérience utilisateur.

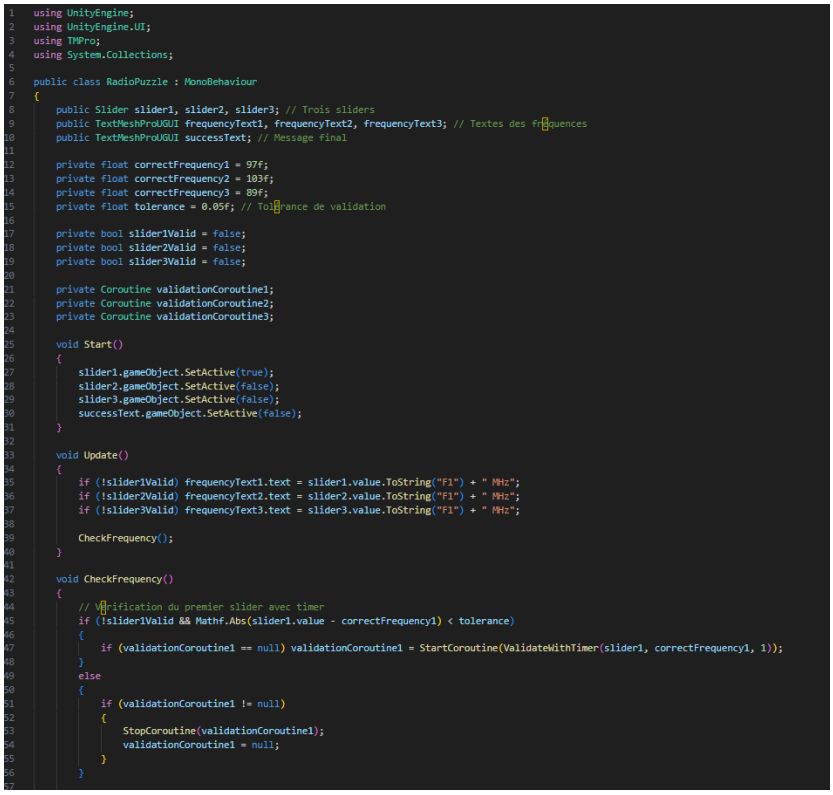
```
public class Keypad : MonoBehaviour
{
    public TMP_InputField charHolder;
    public GameObject button1;
    public GameObject button2;
    public GameObject button3;
    public GameObject button4;
    public GameObject button5;
    public GameObject button6;
    public GameObject button7;
    public GameObject button8;
    public GameObject button9;
    public GameObject button0;
    public GameObject buttonclear;
    public GameObject buttonenter;
    public string password = "5679";
    public void B1(){
        charHolder.text = charHolder.text +"1";
    }
    public void B2(){
        charHolder.text = charHolder.text +"2";
    }
    public void B3(){
        charHolder.text = charHolder.text +"3";
    }
    public void B4(){
        charHolder.text = charHolder.text +"4";
    }
    public void B5(){
        charHolder.text = charHolder.text +"5";
    }
    public void B6(){
        charHolder.text = charHolder.text +"6";
    }
    public void B7(){
        charHolder.text = charHolder.text +"7";
    }
    public void B8(){
        charHolder.text = charHolder.text +"8";
    }
    public void B9(){
        charHolder.text = charHolder.text +"9";
    }
    public void B0(){
        charHolder.text = charHolder.text +"0";
    }
    public void ClearEvent(){
        charHolder.text = null;
    }
    public void enterEvent(){
        if (charHolder.text == password){
            charHolder.text = "Yes";
        }
        else {
            charHolder.text = "No";
        }
    }
}
```

FIGURE 23 – CodeLock



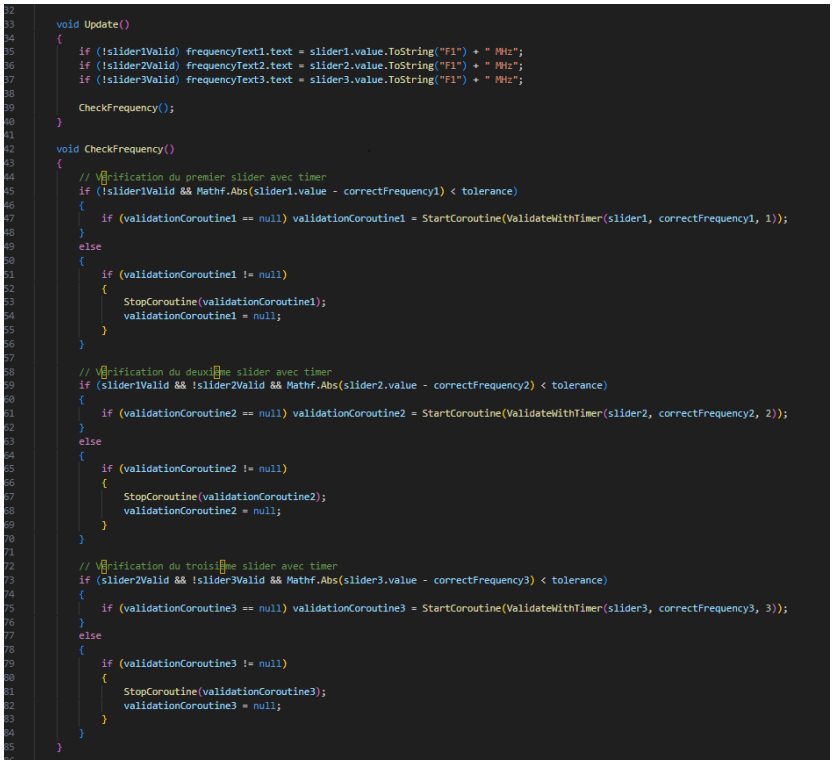
```
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using System.Collections;

public class BathroomPuzzle : MonoBehaviour
{
    public Slider tempSlider;
    public TextMeshProUGUI tempText;
    public GameObject hiddenSymbol; // Image cachée

    private float correctTemperature = 42.0f; // Température correcte
    private float tolerance = 1.0f; // Marge d'erreur
    private Coroutine revealCoroutine;

    void Start()
    {
        hiddenSymbol.SetActive(false); // Cache le symbole au début
    }

    void Update()
    {
        tempText.text = "Température : " + tempSlider.value.ToString("F1") + "°C";
        CheckTemperature();
    }

    void CheckTemperature()
    {
        if (Mathf.Abs(tempSlider.value - correctTemperature) < tolerance)
        {
            if (revealCoroutine == null) revealCoroutine = StartCoroutine(RevealSymbol());
        }
        else
        {
            if (revealCoroutine != null)
            {
                StopCoroutine(revealCoroutine);
                revealCoroutine = null;
                hiddenSymbol.SetActive(false); // Cache le symbole si la température change
            }
        }
    }

    IEnumerator RevealSymbol()
    {
        yield return new WaitForSeconds(2f); // Attendre 2 secondes
        hiddenSymbol.SetActive(true); // Afficher le symbole
    }
}

```

FIGURE 24 – Salle de Bain



```
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using System.Collections;

public class RadioPuzzle : MonoBehaviour
{
    public Slider slider1, slider2, slider3; // Trois sliders
    public TextMeshProUGUI frequencyText1, frequencyText2, frequencyText3; // Textes des fréquences
    public TextMeshProUGUI successText; // Message final

    private float correctFrequency1 = 97f;
    private float correctFrequency2 = 103f;
    private float correctFrequency3 = 89f;
    private float tolerance = 0.05f; // Tolérance de validation

    private bool slider1Valid = false;
    private bool slider2Valid = false;
    private bool slider3Valid = false;

    private Coroutine validationCoroutine1;
    private Coroutine validationCoroutine2;
    private Coroutine validationCoroutine3;

    void Start()
    {
        slider1.gameObject.SetActive(true);
        slider2.gameObject.SetActive(false);
        slider3.gameObject.SetActive(false);
        successText.gameObject.SetActive(false);
    }

    void Update()
    {
        if (!slider1Valid) frequencyText1.text = slider1.value.ToString("F1") + " MHz";
        if (!slider2Valid) frequencyText2.text = slider2.value.ToString("F1") + " MHz";
        if (!slider3Valid) frequencyText3.text = slider3.value.ToString("F1") + " MHz";

        CheckFrequency();
    }

    void CheckFrequency()
    {
        // Vérification du premier slider avec timer
        if (!slider1Valid && Mathf.Abs(slider1.value - correctFrequency1) < tolerance)
        {
            if (validationCoroutine1 == null) validationCoroutine1 = StartCoroutine(ValidateWithTimer(slider1, correctFrequency1, 1));
        }
        else
        {
            if (validationCoroutine1 != null)
            {
                StopCoroutine(validationCoroutine1);
                validationCoroutine1 = null;
            }
        }

```

Figure 25 – Radio.1



```
    void Update()
    {
        if (!slider1Valid) frequencyText1.text = slider1.value.ToString("F1") + " MHz";
        if (!slider2Valid) frequencyText2.text = slider2.value.ToString("F1") + " MHz";
        if (!slider3Valid) frequencyText3.text = slider3.value.ToString("F1") + " MHz";

        CheckFrequency();
    }

    void CheckFrequency()
    {
        // Vérification du premier slider avec timer
        if (!slider1Valid && Mathf.Abs(slider1.value - correctFrequency1) < tolerance)
        {
            if (validationCoroutine1 == null) validationCoroutine1 = StartCoroutine(ValidateWithTimer(slider1, correctFrequency1, 1));
        }
        else
        {
            if (validationCoroutine1 != null)
            {
                StopCoroutine(validationCoroutine1);
                validationCoroutine1 = null;
            }
        }

        // Vérification du deuxième slider avec timer
        if (slider1Valid && !slider2Valid && Mathf.Abs(slider2.value - correctFrequency2) < tolerance)
        {
            if (validationCoroutine2 == null) validationCoroutine2 = StartCoroutine(ValidateWithTimer(slider2, correctFrequency2, 2));
        }
        else
        {
            if (validationCoroutine2 != null)
            {
                StopCoroutine(validationCoroutine2);
                validationCoroutine2 = null;
            }
        }

        // Vérification du troisième slider avec timer
        if (slider2Valid && !slider3Valid && Mathf.Abs(slider3.value - correctFrequency3) < tolerance)
        {
            if (validationCoroutine3 == null) validationCoroutine3 = StartCoroutine(ValidateWithTimer(slider3, correctFrequency3, 3));
        }
        else
        {
            if (validationCoroutine3 != null)
            {
                StopCoroutine(validationCoroutine3);
                validationCoroutine3 = null;
            }
        }
    }
```

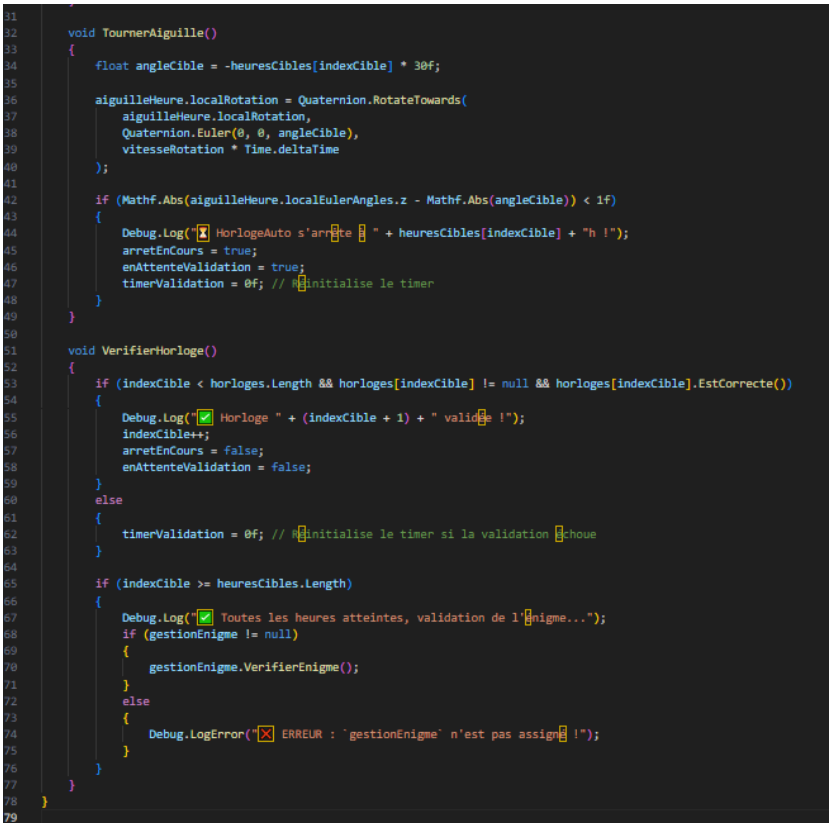
FIGURE 26 – Radio.2

```
86
87      IEnumerator ValidateWithTimer(Slider slider, float correctValue, int sliderNumber)
88      {
89          float timeLeft = 3f;
90
91          while (timeLeft > 0)
92          {
93              // Si le joueur bouge la fréquence, on annule le timer
94              if (Mathf.Abs(slider.value - correctValue) > tolerance)
95              {
96                  yield break;
97              }
98
99              yield return new WaitForSeconds(1f);
100             timeLeft -= 1f;
101         }
102
103         // Si le joueur reste 3 secondes sur la bonne fréquence, on valide le slider
104         slider.value = correctValue;
105         slider.interactable = false;
106
107         if (sliderNumber == 1)
108         {
109             slider1Valid = true;
110             slider2.gameObject.SetActive(true); // Débloquer Slider 2
111         }
112         else if (sliderNumber == 2)
113         {
114             slider2Valid = true;
115             slider3.gameObject.SetActive(true); // Débloquer Slider 3
116         }
117         else if (sliderNumber == 3)
118         {
119             slider3Valid = true;
120             successText.gameObject.SetActive(true); // Afficher le message final
121             successText.text = "🎉 Félicitations ! Vous avez trouvé toutes les fréquences ! 🎉";
122         }
123     }
124 }
```

FIGURE 27 – Radio.3

```
1   using UnityEngine;
2
3   public class HorlogeAuto : MonoBehaviour
4   {
5       public Transform aiguilleHeure;
6       public float vitesseRotation = 30f;
7       public int[] heuresCibles = { 3, 6, 9 };
8       private int indexCible = 0;
9       private bool arretEnCours = false;
10      private float timerValidation = 0f;
11      private bool enAttenteValidation = false;
12
13      public Horloge[] horloges;
14      public GestionEnigme gestionEnigme; // Vérifie que ceci est bien assigné dans l'Inspector
15
16      void Update()
17      {
18          if (!arretEnCours && indexCible < heuresCibles.Length)
19          {
20              TournerAiguille();
21          }
22          else if (enAttenteValidation)
23          {
24              timerValidation += Time.deltaTime;
25              if (timerValidation >= 3f)
26              {
27                  VerifierHorloge();
28              }
29          }
30      }
31
```

FIGURE 28 – HorlogeBroken.1



```
    void TournerAiguille()
    {
        float angleCible = -heuresCibles[indexCible] * 30f;

        aiguilleHeure.localRotation = Quaternion.RotateTowards(
            aiguilleHeure.localRotation,
            Quaternion.Euler(0, 0, angleCible),
            vitesseRotation * Time.deltaTime
        );

        if (Mathf.Abs(aiguilleHeure.localEulerAngles.z - Mathf.Abs(angleCible)) < 1f)
        {
            Debug.Log("⏳ HorlogeAuto s'arrête à " + heuresCibles[indexCible] + "h !");
            arretEnCours = true;
            enAttenteValidation = true;
            timerValidation = 0f; // Réinitialise le timer
        }
    }

    void VerifierHorloge()
    {
        if (indexCible < horloges.Length && horloges[indexCible] != null && horloges[indexCible].EstCorrecte())
        {
            Debug.Log("✅ Horloge " + (indexCible + 1) + " validée !");
            indexCible++;
            arretEnCours = false;
            enAttenteValidation = false;
        }
        else
        {
            timerValidation = 0f; // Réinitialise le timer si la validation échoue
        }

        if (indexCible >= heuresCibles.Length)
        {
            Debug.Log("✅ Toutes les heures atteintes, validation de l'énigme...");
            if (gestionEnigme != null)
            {
                gestionEnigme.VerifierEnigme();
            }
            else
            {
                Debug.LogError("❌ ERREUR : `gestionEnigme` n'est pas assigné !");
            }
        }
    }
}
```

FIGURE 29 – HorlogeBroken.2

```
using UnityEngine;
using System.Collections;

public class Horloge : MonoBehaviour
{
    public Transform aiguilleHeure; // Aiguille propre à cette horloge
    private int heureActuelle = 12; // Heure affichée
    public int heureCorrecte; // Heure correcte pour résoudre l'énigme
    public bool estActive = false; // Seule une horloge active peut être modifiée
    public Horloge horlogeSuivante; // Référence vers la prochaine horloge
    private bool joueurProche = false; // Détection du joueur


    void Start()
    {
        Debug.Log(gameObject.name + " estActive = " + estActive);
    }
    void Update()
    {

        if (estActive) // Supprimé le check joueurProche pour tester
        {
            if (Input.GetKeyDown(KeyCode.RightArrow))
            {
                Debug.Log("➡ Flèche Droite détectée !");
                ChangerHeure(1);
            }
            if (Input.GetKeyDown(KeyCode.LeftArrow))
            {
                Debug.Log("⬅ Flèche Gauche détectée !");
                ChangerHeure(-1);
            }
        }

    }

    public void ChangerHeure(int increment)
    {
        heureActuelle = (heureActuelle + increment) % 12;
        if (heureActuelle == 0) heureActuelle = 12;
        float rotationAngle = (360f / 12) * heureActuelle;
        aiguilleHeure.localRotation = Quaternion.Euler(0, 0, -rotationAngle);

        if (EstCorrecte())
        {
            Debug.Log(gameObject.name + " bien réglée !");
            ActiverHorlogeSuivante();
        }

        if (!estActive)
        {
            Debug.Log("❌ L'horloge " + gameObject.name + " est désactivée !");
            return;
        }

        Debug.Log("⏳ L'horloge " + gameObject.name + " réglée sur " + heureActuelle + "h");
    }

    IEnumerator AttendreValidation()
    {
        yield return new WaitForSeconds(3f);
        ActiverHorlogeSuivante();
    }

    public bool EstCorrecte()
    {
        return heureActuelle == heureCorrecte;
    }

```

FIGURE 30 – Horloge.1

```
    public bool EstCorrecte()
    {
        return heureActuelle == heureCorrecte;
    }

    void ActiverHorlogeSuivante()
    {
        estActive = false;
        if (horlogeSuivante != null)
        {
            horlogeSuivante.estActive = true;
            Debug.Log(horlogeSuivante.gameObject.name + " est maintenant active !");
        }
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            joueurProche = true;
            Debug.Log("✅ Joueur proche de " + gameObject.name);
        }
    }

    private void OnTriggerExit2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            joueurProche = false;
            Debug.Log("❌ Joueur éloigné de " + gameObject.name);
        }
    }
}
```

FIGURE 31 – Horloge.2

```
using UnityEngine;
using TMPro;

public class CassettePuzzle : MonoBehaviour
{
    public TextMeshProUGUI cassetteText; // Texte affichant l'indice

    // Liste des indices correspondant aux cassettes
    private string[] cassetteMessages = {
        "Indice 1 : Cherche dans l'ombre...",
        "Indice 2 : La clé est sous la table...",
        "Indice 3 : Le code est 482"
    };

    // Fonction pour insérer une cassette
    public void InsertCassette(int cassetteNumber)
    {
        if (cassetteNumber >= 0 && cassetteNumber < cassetteMessages.Length)
        {
            cassetteText.text = cassetteMessages[cassetteNumber]; // Afficher l'indice
        }
    }
}
```

FIGURE 32 – Cassette

3.3 Retards

— **Multijoueur**

Nous avons rencontré des difficultés lors de la création de scènes simultanées, où les personnages devaient se trouver dans des scènes différentes. En effet, Netcode for GameObjects n'est initialement pas conçu pour gérer ce type de scénario, ce qui a nécessité une personnalisation de la gestion des scènes. Après plusieurs tentatives infructueuses et de nombreux échecs, nous avons pris du retard dans notre développement. En conséquence, nous avons pris la décision de simplifier la gestion en réunissant toutes les pièces dans une seule scène, ce qui nous permet de mieux gérer la synchronisation des personnages et des éléments du jeu en multijoueur. Cette approche a permis de réduire la complexité technique tout en maintenant la fonctionnalité du jeu.

— **Enigmes**

L'équipe a rencontré des difficultés lors de l'implémentation des énigmes, celles-ci ayant été conçues en amont mais posant des problèmes d'intégration dans le projet. Malgré nos efforts pour les adapter, certaines mécaniques ne fonctionnaient pas comme prévu, nécessitant des ajustements et une réévaluation de leur mise en place.

4 Plan d'action pour la prochaine soutenance

Dans le cadre de la prochaine soutenance, nous avons élaboré un plan d'action rigoureux visant à assurer des progrès significatifs dans le développement de notre jeu. Ce plan repose sur les points suivants :

— **Multijoueur :**
NOus aurons régler les soucis de séparation des joueurs.

— **Musique :**
Il n'y a plus qu'à implémenter la musique dans toutes les scènes, ainsi que trouver une musique pour le combat contre le boss.

— **Enigmes :**
Toutes les énigmes seront prêtes afin de ralentir nos joueurs dans leur progression.

— **Ennemis et IA :**
La conception des ennemis et de l'intelligence artificielle seront terminés. Ces derniers seront prêt à faire face aux joueurs.

— **Map :**
Il ne manque plus que le rez-de-chaussée et la salle du boss.

Ces étapes représentent des progrès importants dans notre processus de développement, et nous sommes impatients de partager ces avancées lors de la prochaine soutenance.

5 Annexes

Nous avons mis en place des réseaux sociaux ou vous pouvez nous retrouver et suivre les avancées du jeu Shattered Mind.

— **Discord :**
discord.gg/RFQ5CZMFJq*

— **L'instagram :**
@devonkhain

— **Linkedin :**
www.linkedin.com/in/devonkhain-inc-b4a4ab340

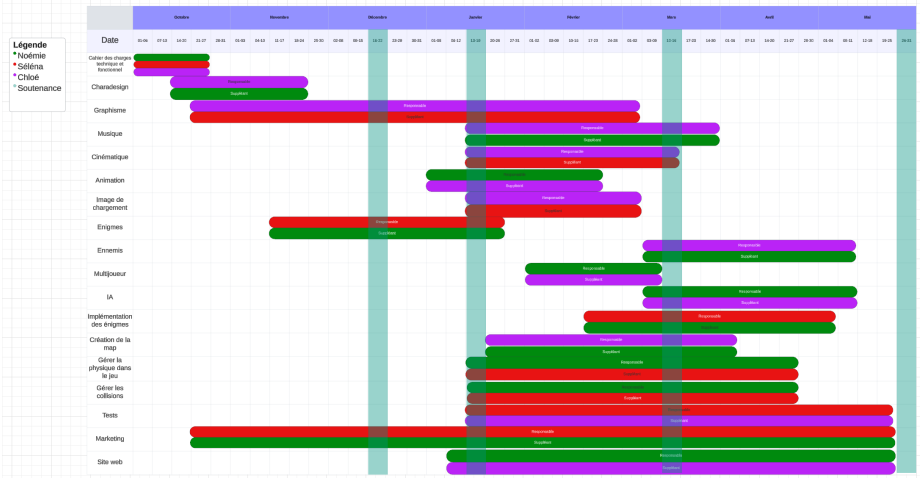


FIGURE 33 – Diagramme de Gantt