

Rapport de Projet

LA GRANDE EVASION

Groupe 9 ("Epi Games")

18 juin 2024



AKHDAR Wael (chef de groupe)

ZAHRA Youcef

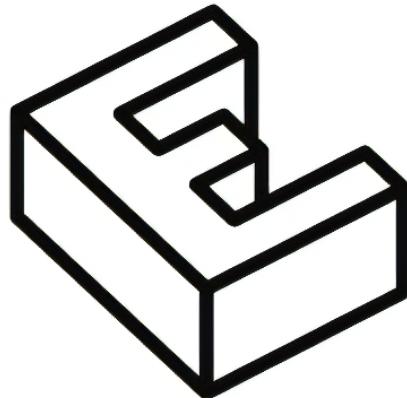
ZIANI Jessim

OLLER Ilann

LASSIK Nabil

Résumé

Notre entreprise “Epi Games” est composée de cinq élèves de première année à EPITA. Chaque membre mettra en œuvre ses compétences afin de concevoir un jeu vidéo d’escape game en trois dimensions, nommée LA GRANDE ÉVASION. Nous réalisons ce projet dans le but de mettre en œuvre nos connaissances acquises durant cette première année en école d’ingénieur, et de découvrir de nouveaux domaines tels que : l’Intelligence Artificielle, la modélisation 3D, la connectivité réseau, le développement Web.



EPI GAMES

FIGURE 1 – Logo Epi Games

Table des matières

1	Introduction	5
2	Retour sur le cahier des charges	6
2.1	Scénario	6
2.2	Tableau caractéristique du jeu	7
2.3	Diagramme de Gantt	8
3	Avancement du jeu avant la première soutenance technique	9
3.1	Design du jeu	9
3.1.1	Environnement	9
3.1.2	Personnage	11
3.1.3	Garde	11
3.2	Mécanique de jeu	12
3.2.1	Mouvements du joueur	12
3.2.2	Animations du joueur	15
3.3	Intelligence Artificielle (IA)	19
3.3.1	Navmesh	19
3.3.2	Mouvement IA	21
3.4	Site Web	22
3.4.1	Architecture du site Web	22
3.4.2	Sections du site Web	22
4	Prévisions de l'avancement après la soutenance	24
4.1	Design du jeu	24
4.1.1	Prison	24
4.1.2	Personnage	25
4.1.3	Garde	26
4.1.4	Objets	26
4.1.5	Menu du jeu	26

4.2	Mécanique de jeu	26
4.2.1	Inventaire	27
4.3	Intelligence Artificielle (IA)	28
4.3.1	Mouvement IA	28
4.3.2	Animation IA	28
4.4	Mode multijoueur	28
4.5	Site Web	28
5	Avancement après la première soutenance technique	29
5.1	Design du jeu	29
5.1.1	Prison	29
5.1.2	Personnage	31
5.1.3	Garde	33
5.1.4	Objets	35
5.1.5	Menu du jeu	35
5.1.6	Option	37
5.1.7	Interfaces de victoire et de défaite	38
5.2	Mécaniques du jeu	40
5.2.1	Mouvements du joueur	40
5.2.2	Animations du joueur	42
5.2.3	Inventaire	43
5.2.4	Énigmes	45
5.2.5	Vies	49
5.2.6	Sons	50
5.3	Intelligence Artificielle	52
5.3.1	Mouvements	52
5.3.2	Animations	58
5.3.3	Représentation du champ de vision et du son	60
5.4	Mode Multijoueur	61
5.5	Modification des logos	65

5.5.1	Comparaison ancien et nouveau logo	65
5.5.2	Polyvalence du nouveau logo	67
5.6	Site Web	69
6	Conclusion	70

1 Introduction

Epi Games est heureux de présenter son jeu "LA GRANDE ÉVASION". Ce jeu est un jeu d'escape game en 3D se déroulant dans un environnement carcéral. L'objectif est de réussir à s'échapper. Pour y parvenir, la collaboration entre les joueurs est essentielle : ils devront résoudre une série d'énigmes et échapper à la surveillance du garde pour retrouver leurs libertés. Après avoir acquis des compétences approfondies dans l'utilisation de la plateforme de développement Unity, nous avons atteint nos objectifs comme nous l'avions prévu.



FIGURE 2 – Logo La Grande Evasion

2 Retour sur le cahier des charges

2.1 Scénario

Quatre amis se retrouvent capturés et enfermés par un docteur obsédé par l'idée de les utiliser dans des expériences totalement folles. Leur objectif est donc de s'évader. Les prisonniers explorent méthodiquement la prison, exploitant la moindre faille de sécurité pour espérer s'échapper. Durant leur tentative d'évasion, le docteur patrouille activement dans la prison, traquant sans répit ceux qui oseraient s'échapper. S'il réussit à les attraper, il les ramène à leur cellule.

2.2 Tableau caractéristique du jeu

Type de jeu :				
Action/Aventure	Battle Royale	Beat them all	Combat	Simulation
FPS	MMORPG	MOBA	Party Games	Survival Horror
Plateforme	Puzzles	Reflexion	Rogue Like	TPS
RPG	RTS	Sandbox	Shoot them up	Course
Autre :				
Caractéristiques générales du jeu :				
IA : Errer	Attaquer	S'échapper	"Path Finder"	
Multijoueurs : Coopé	Battle (2-4)	Massif		
Réseau : P2P	Lan	Online		
Caractéristiques graphiques :				
Dimension : 2D	3D	Autres :		
Particularités : Stéréoscopie	AR	VR		
graphiques : Perso	Custom	Existant		
Précisions :				
Caractéristiques sonores :				
Musique : Perso	Custom	Existant		
FX : Perso	Custom	Existant		
Précisions :				
Autres caractéristiques :				
Site Web : Perso	Custom	Préfabriqué		

FIGURE 3 – Ancien cahier des charges technique

Type de jeu :				
Action/Aventure	Battle Royale	Beat them all	Combat	Simulation
FPS	MMORPG	MOBA	Party Games	Survival Horror
Plateforme	Puzzles	Reflexion	Rogue Like	TPS
RPG	RTS	Sandbox	Shoot them up	Course
Autre :				
Caractéristiques générales du jeu :				
IA : Errer	Attaquer	S'échapper	"Path Finder"	
Multijoueurs : Coopé	Battle (2-4)	Massif		
Réseau : P2P	Lan	Online		
Caractéristiques graphiques :				
Dimension : 2D	3D	Autres :		
Particularités : Stéréoscopie	AR	VR		
graphiques : Perso	Custom	Existant		
Précisions :				
Caractéristiques sonores :				
Musique : Perso	Custom	Existant		
FX : Perso	Custom	Existant		
Précisions :				
Autres caractéristiques :				
Site Web : Perso	Custom	Préfabriqué		

FIGURE 4 – Nouveau cahier des charges technique

La mise en place du mode multijoueur est un élément clé de notre projet, et après mûre réflexion, nous avions décidé d'opter pour un mode multijoueur en ligne plutôt qu'en local (LAN). Cette décision découle de notre volonté à rendre l'expérience de jeu plus accessible et conviviale. Le choix du mode multijoueur en ligne permettra aux joueurs de se connecter et de jouer ensemble sur différents appareils, peu importe leur proximité physique, qui était un facteur clé pour le mode LAN.

2.3 Diagramme de Gantt

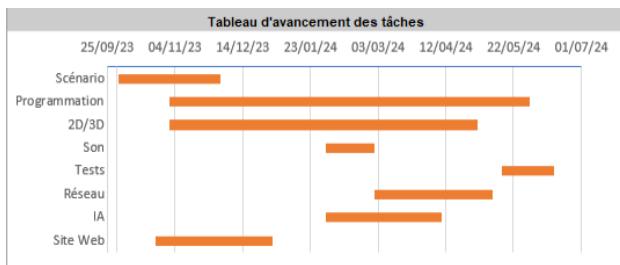


FIGURE 5 – Ancien diagramme de Gantt

Nous avons réalisé que les tests sont une étape cruciale qui s'étend tout au long du processus de création du jeu, ce qui nous a amenés à prolonger cette phase. De plus comme énoncé précédemment, nous avons remis en question le type de réseau utilisé et donc retardé sa conception, de ce fait nous avons donc priorisé l'avancement de l'IA afin de prévenir tout retard éventuel. Le scénario et le site web ont été effectués dans les temps. Concernant les aspects sonores et visuels du jeu, nous avions décidé de les reporter temporairement afin de concentrer nos efforts sur le développement de l'intelligence artificielle. Cela ne signifie pas un abandon complet du son, mais plutôt une réorganisation des tâches. Nous croyons fermement que cette approche nous permettra de garantir la fonctionnalité du jeu, même si cela implique de sacrifier certains détails esthétiques à ce moment-là.



FIGURE 6 – Nouveau diagramme de Gantt

3 Avancement du jeu avant la première soutenance technique

3.1 Design du jeu

3.1.1 Environnement

Le jeu se déroule dans une prison, nous devons donc la concevoir en 3D, elle aura deux niveaux : les cellules en haut et la salle principale en bas.

Nous avons opté pour Blender, un logiciel de modélisation et d'importation d'éléments 3D, pour cette tâche. Pour obtenir les éléments nécessaires, nous avons utilisé différentes ressources, principalement sur le site Sketchfab, qui propose une grande variété d'éléments 3D. Nous avons trouvé une représentation de prison qui correspondait approximativement à nos besoins, que nous avons ensuite modifié pour l'adapter parfaitement à notre vision du jeu.

Les modifications ont porté sur l'intérieur et l'extérieur de la prison. À l'intérieur, nous avons supprimé des cellules du niveau inférieur pour libérer de l'espace pour la salle principale, nous avons aussi ajouté des tables, réduit le nombre de cellules au niveau supérieur pour qu'il y en ait quatre, comme le nombre de joueurs maximum dans le cas d'une partie multijoueur. De plus, nous avons entièrement modifié le décor des cellules, y compris les lits, les toilettes et les lavabos.



FIGURE 7 – Avant-Après de l'intérieur de la prison

Concernant l'extérieur de la prison, nous avons corrigé certains problèmes esthétiques du modèle récupéré sur Sketchfab, tels que des barreaux et le sol dépassant des murs. Nous nous sommes rendus compte seulement plus tard que l'esthétique extérieur de la prison n'était d'aucune utilité sachant qu'aucun utilisateur ne pourrait le voir.



FIGURE 8 – Avant-Après de l'extérieur de la prison

Pour les textures, nous avons utilisé celles des modèles 3D récupérés sur Sketchfab.

3.1.2 Personnage

Pour le personnage principal nous n'avions pas encore de design définitif. En revanche, nous utilisions momentanément un personnage 3D nommé "Y Bot" accessible sur la plateforme Mixamo, représentant un robot pour implémenter les mouvements.

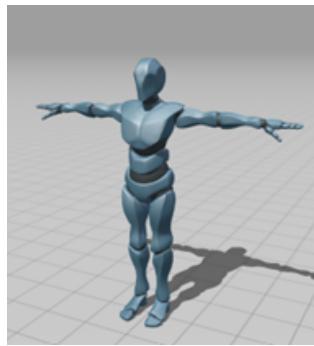


FIGURE 9 – Image de Ybot

3.1.3 Garde

Pour le design du garde de la prison, nous avions un design qui a finalement été modifié par la suite. Nous utilisions pour momentanément un personnage 3D accessible sur l'asset store de Unity représentant un garde de prison.



FIGURE 10 – Image du garde

3.2 Mécanique de jeu

3.2.1 Mouvements du joueur

Pour les mouvements du joueur, nous avons créé un script permettant au personnage de se déplacer verticalement et horizontalement :

```
float verticalAxis = Input.GetAxis("Vertical");
float horizontalAxis = Input.GetAxis("Horizontal");
```

FIGURE 11 – Variables des axes

Dans un premier temps nous avons récupéré l'entrée de l'axe vertical et de l'axe horizontal et les avons assignés respectivement aux variables verticalAxis et horizontalAxis.

```
else if(verticalAxis>0)
{
    PlayerAnimator.SetFloat("walk vertical", 1);
    Vector3 movement = this.transform.forward;
    movement.Normalize();
    rb.MovePosition(transform.position + movement*speed*time.fixedDeltaTime);
}
```

FIGURE 12 – Déplacements vers l'avant

Ensuite, si l'axe vertical est supérieur à 1, autrement dit si la touche “Flèche du haut” est enfoncé, un vecteur “movement” est créé représentant la direction vers l' avant du joueur , puis nous généralisons ce vecteur afin de garantir une vitesse constante, enfin nous déplaçons le Rigidbody du joueur vers l'avant en ajoutant movement au produit de la vitesse de déplacement (speed) et du temps écoulé depuis la dernière frame (Time.fixedDeltaTime) à la position actuelle du joueur (transform.position).

Cette condition est répétée pour le cas où l'axe vertical est inférieur à -1 à la seule différence que le vecteur "movement" est opposé, de plus, la variable de vitesse de déplacement "speeder" est inférieure a "speed" :

```
else if(verticalAxis<0)
{
    PlayerAnimator.SetFloat("walk vertical", -1);
    Vector3 movement = this.transform.forward * verticalAxis;
    movement.Normalize();
    rb.MovePosition(transform.position + movement*speedar*Time.fixedDeltaTime);
}
```

FIGURE 13 – Déplacements vers l'arrière

Nous avons procédé de la même manière pour les mouvements horizontaux, sauf que le vecteur assigné à la variable "movement" est transform.right (représentant le vecteur pointant vers la droite du joueur) et ici "speedcote" est égal a "speed" :

```
else if(horizontalAxis>0)
{
    PlayerAnimator.SetFloat("walk horizontal", 1);
    Vector3 movement = this.transform.right;
    movement.Normalize();
    rb.MovePosition(transform.position + movement*speedcote*Time.fixedDeltaTime);
}
else if(horizontalAxis<0 )
{
    PlayerAnimator.SetFloat("walk horizontal", -1);
    Vector3 movement = -this.transform.right;
    movement.Normalize();
    rb.MovePosition(transform.position + movement*speedcote*Time.fixedDeltaTime);
}
```

FIGURE 14 – Déplacements vers la droite et vers la gauche

De plus nous avons créé une condition permettant au joueur d'accélérer (sprint) :

```
if (Input.GetKey(KeyCode.LeftShift) && verticalAxis>0 )
{
    PlayerAnimator.SetFloat("run", 1);
    PlayerAnimator.SetFloat("walk vertical", 1);
    Vector3 movement = this.transform.forward * verticalAxis;
    movement.Normalize();
    rb.MovePosition(transform.position + movement*speedf*Time.fixedDeltaTime);
}
```

FIGURE 15 – Sprint

Si les touches “flèche du haut” et “maj gauche” sont enfoncées, le même code est appliqué que celui pour se déplacer vers l'avant sauf que la vitesse de déplacement est plus importante (speedf > speed).

Pour finir nous avons ajouté un script de saut :

```
bool canJump = Time.time - lastJumpTime >= 1.5f;
```

FIGURE 16 – Condition pour sauter

```
if (Input.GetButton("Jump") && canJump )
{
    rb.AddForce(Vector3.up*4,ForceMode.Impulse);
    PlayerAnimator.SetFloat("jump", 1);
    lastJumpTime = Time.time;
```

FIGURE 17 – Saut

Ce script crée une variable booléenne qui renvoie si le joueur peut sauter, (le joueur peut sauter chaque 1.5 secondes), si le bouton espace est enfoncé et que le joueur peut sauter, nous ajoutons une force au rigidbody du joueur, la force est dans la direction “Vector3.up” (vers le haut) multipliée par 4 unités, et appliquée avec “ForceMode.Impulse”, signifiant qu'elle est appliquée une seule fois. Enfin, la dernière ligne enregistre le moment où le

saut a eu lieu en stockant la valeur actuelle du temps dans la variable lastJumpTime. Cela sera utile pour limiter le nombre de saut.

3.2.2 Animations du joueur

Nous avons amélioré l'expérience de jeu en introduisant des animations au joueur, activées dans des circonstances spécifiques. Une animation de respiration se déclenche lorsque le joueur est immobile :

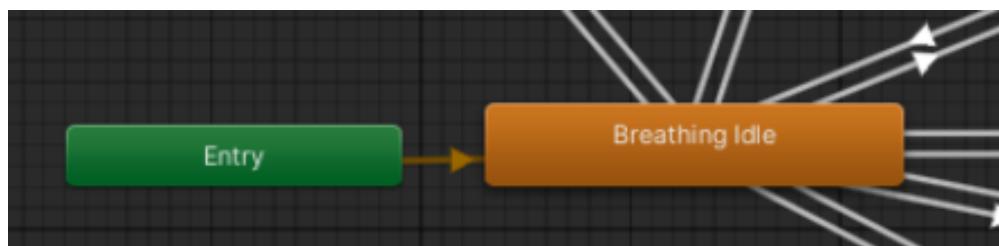


FIGURE 18 – Animation par défaut

Pour faire ceci, après avoir importé l'animation depuis le site Mixamo, nous avons ajouté une transition du bloc "Entry" à notre animation "Breathing Idle", permettant donc à cette animation d'être activée par défaut sur notre personnage.

Pour les autres animations nous avons dû créer des variables et des conditions sur celles-ci :

Ces quatre variables sont des nombres réels (float) et les conditions associées à ces variables sont celles-ci :

1. **Walking** : si la variable "walk vertical" est supérieure à 0.1 et que "run" est inférieur à 0.1, alors l'animation de marche est déclenchée. et si "walk vertical" est inférieur à 0.1 l'animation repasse en "Breathing Idle" (animation par défaut).

= walk vertical	0
= walk horizontal	0
= jump	0
= run	0

FIGURE 19 – Variables de l'animator

Conditions

= walk vertical	Greater	0.1
= run	Less	0.1

(a) Condition de démarrage "Walking"

Conditions

= walk vertical	Less	0.1
-----------------	------	-----

(b) Condition d'arrêt "Walking"

FIGURE 20 – Illustration des conditions de démarrage et d'arrêt "Walking"

2. **Left/Right Strafe Walking** : même principe, mais avec la variable "walk horizontal" pour se diriger vers la droite et la gauche.

3. **Walking Backwards** : similaire au fonctionnement des autres animations.

4. **Jumping** : déclenché en fonction de la condition de saut.

5. **Run** : cette animation d'accélération peut être activée à deux moments, donc elle possède deux conditions de démarrage et deux conditions d'arrêt, contrairement aux autres animations. Ces conditions sont également basées sur les mouvements du personnage.

Conditions

= run	Greater	0.1
= walk vertical	Greater	0.1

(a) 1ère condition de démarrage "Run"

Conditions

= run	Less	0.1
-------	------	-----

(b) 1ère condition d'arrêt "Run"

FIGURE 21 – Illustration des conditions de démarrage et d'arrêt "Run"

Toutes ces conditions sont implémentées dans des transitions partant de l'état "Breathing Idle" dans l'Animator.

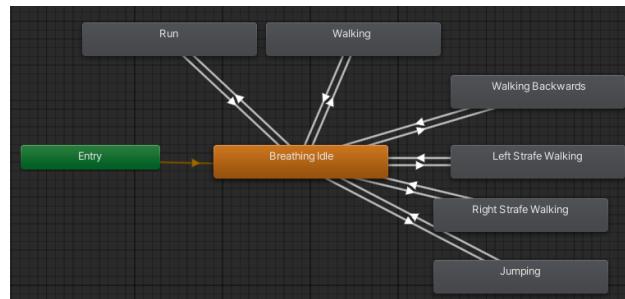


FIGURE 22 – Animator de “Y Bot”

Sauf ces deux dernières permettant de passer de l'état "Walking" à "Run" et vice-versa :

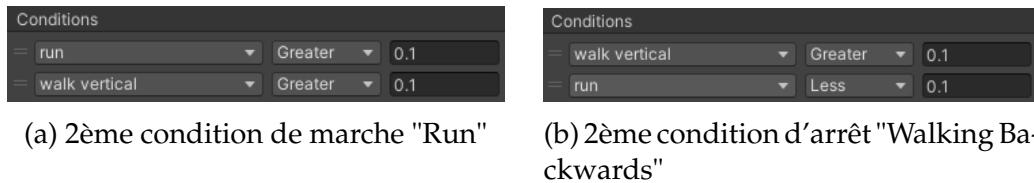


FIGURE 23 – Illustration des deux conditions

Pour illustrer, l'Animator final prend la forme suivante :

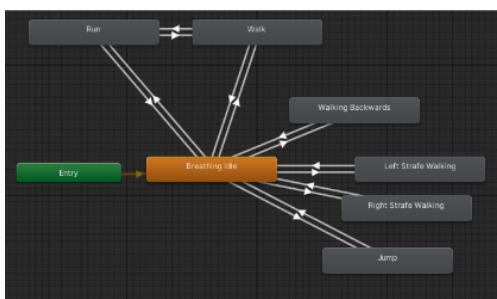


FIGURE 24 – Animator Final de “Y Bot”

Il est également important de noter que les variables mentionnées précédemment sont modifiées dans le script move.cs (figure 11-17). Par exemple, la variable "Jump" est initialisée à 0 au début du script et est ensuite réglée à 1 lorsque la condition de saut est vérifiée.

```
PlayerAnimator.SetFloat("jump", 0);
```

(a) Initialisation de "jump"

```
PlayerAnimator.SetFloat("jump", 1);
```

(b) Incrémentation de "jump"

FIGURE 25 – Illustrations de l'initialisation et de l'incrémentation de "jump"

3.3 Intelligence Artificielle (IA)

3.3.1 Navmesh

Nous avons appliqué le Nav Mesh sur le sol de la prison, suivi de l'application de modificateurs de Nav Mesh aux piliers de la salle dans le but d'augmenter la surface du Nav Mesh.

Le Nav Mesh, ou maillage de navigation, est une représentation tridimensionnelle de l'environnement de jeu. Il est utilisé par les agents de navigation pour se déplacer efficacement, en divisant l'espace en régions accessibles et inaccessibles.

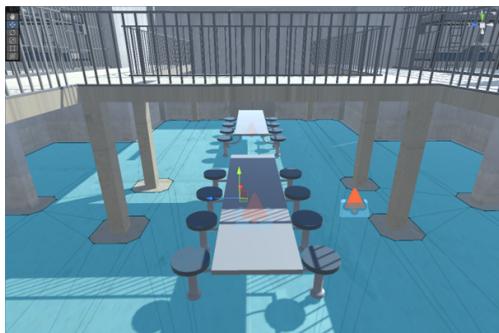


FIGURE 26 – Nav Mesh avant le Nav Mesh Modifier

Quant aux Nav Mesh Modifier, ce sont des composants permettant d'ajuster la surface du Nav Mesh. Dans ce contexte, leur application aux piliers de la salle vise à étendre la zone navigable du Nav Mesh autour des obstacles, comme les piliers. Cela permet aux agents de navigation de contourner ces obstacles de manière fluide tout en explorant l'environnement de jeu.

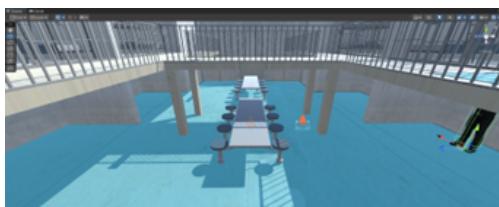


FIGURE 27 – Nav Mesh après le Nav Mesh Modifier

Étant donné que les marches étaient trop courtes pour être correctement franchies par les agents de navigation, malgré nos tentatives de réglage des paramètres tels que la hauteur de marche (step height) ou la pente maximale (max slope), nous avons été contraints d'utiliser un NavMesh Link. Ce dernier permet de créer une connexion entre deux surfaces du NavMesh qui ne sont pas directement accessibles par les algorithmes de navigation standard. En résumé, cela permet aux agents de navigation de passer d'une zone à une autre de manière fluide, même si elles ne sont pas directement reliées sur le Nav Mesh initial.

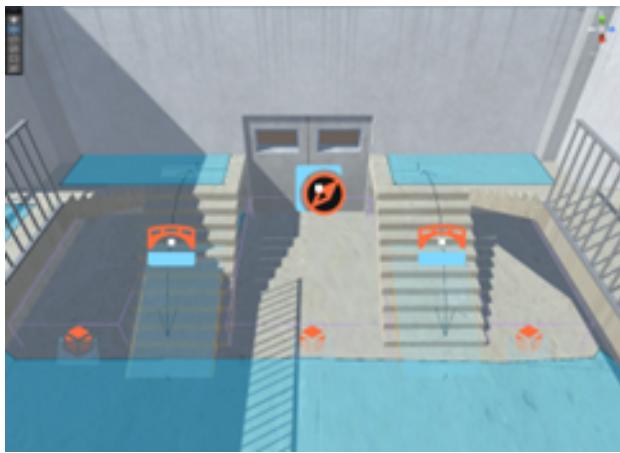


FIGURE 28 – Nav Mesh Link reliant les deux Nav Mesh Surface

3.3.2 Mouvement IA

Le script déplacement.cs est responsable de la navigation du garde dans l'environnement du jeu.

Au démarrage du jeu, la méthode Start() est appelée. Dans cette méthode, le script récupère le composant NavMeshAgent attaché à l'objet (le garde) et l'assigne à la variable agent. Le composant NavMeshAgent est responsable de la navigation du garde dans l'environnement du jeu.

```
void Start()
{
    agent = GetComponent<NavMeshAgent>();
}
```

FIGURE 29 – méthode Start()

À chaque frame, la méthode Update() est appelée. Dans cette méthode, la destination de l'agent de navigation (agent.destination) était mise à jour pour être égale à la position du joueur (YBot.position). Cela permettait au garde de se déplacer automatiquement vers la position du joueur à chaque frame.

```
void Update()
{
    agent.destination = YBot.position;
}
```

FIGURE 30 – méthode Update()

Le code utilise donc le composant NavMeshAgent de Unity pour permettre au garde de suivre le joueur de manière dynamique. Cette approche simplifie la gestion du mouvement du garde, car le système de navigation de Unity se charge de la détection d'obstacles et du calcul des chemins optimaux.

3.4 Site Web

3.4.1 Architecture du site Web

Les carrés symbolisent les pages du site, chacune étant identifiée par son nom situé juste au-dessus. Les flèches placées au-dessus représentent les liens permettant d'accéder à ces pages, tandis que celles situées en-dessous décrivent les diverses rubriques qu'elles contiennent.

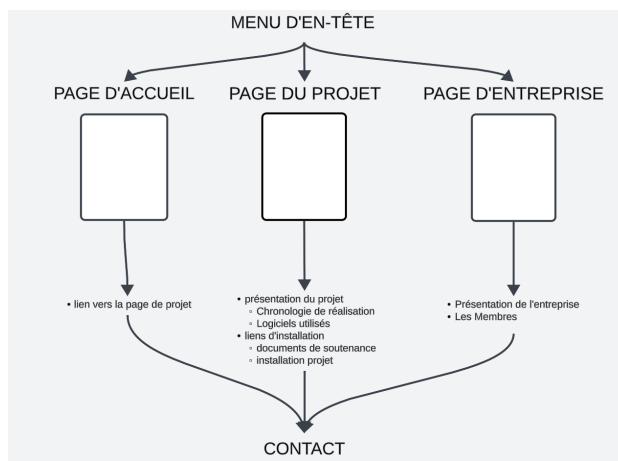


FIGURE 31 – architecture du site Web

3.4.2 Sections du site Web

Afin d'avoir un rendu visuel plus agréable du site Web, nous l'avons mis en forme avec du CSS, un langage de programmation destiné au formatage des pages HTML. La première page fait office de page d'accueil avec notamment la couverture du jeu à la une de notre site, ici La Grande Évasion.

Ensuite vient la page du projet, cette page indique toute information utile sur l'avancement des tâches ou les logiciels utilisés. Il y a aussi des liens de téléchargement du jeu et des documents de soutenance tels que les rapports de soutenances, les plans de soutenances ou encore le rapport de projet. Cependant, le bouton pour le téléchargement du jeu ou pour le rapport de projet ne sont pas encore fonctionnels car le projet n'est pas encore terminé. Sur la dernière page, nous y avons mis les informations plus centrées sur l'entreprise telles que les membres de celle-ci.

Enfin, nous avons un menu d'en-tête et un pied-de-page présents sur toutes les pages. Le menu d'en-tête sert à sélectionner la page souhaitée tandis que le pied-de-page donne des informations utiles pour nous contacter.

4 Prévisions de l'avancement après la soutenance

4.1 Design du jeu

Des ajustements devaient être apportés au design du jeu afin d'optimiser son rendu final. Ces modifications visent à améliorer différents aspects pour garantir une expérience de jeu optimale et immersive.

4.1.1 Prison

Pour la suite du projet, nous devions concentrer nos efforts sur la modification de la prison à l'intérieur pour améliorer le décor, afin qu'il corresponde parfaitement à l'environnement souhaité pour le scénario du jeu. Cette adaptation est essentielle pour créer une atmosphère immersive et cohérente avec l'intrigue. Nous devons concevoir un environnement qui non seulement plonge l'utilisateur dans l'ambiance souhaitée mais qui offre également des éléments propices à l'intégration d'énigmes et de défis pour les joueurs.

En nous basant sur le scénario, nous devions envisager des décors intérieurs qui évoquent à la fois le sentiment d'oppression d'une prison et le contexte de l'expérimentation scientifique menée par le personnage du docteur. Cela pourrait inclure des détails tels que des équipements de laboratoire abandonnés, des graffitis laissés par d'autres prisonniers, des indices sur les expériences menées par le scientifique, etc. Nous devions également veiller à ce que ces éléments décoratifs servent de pistes pour résoudre les énigmes et avancer dans le jeu.

En ce qui concerne l'extérieur de la prison, bien que nous n'ayons pas prévu de modifications spécifiques, nous devons rester ouverts aux ajustements nécessaires. Cela pourrait inclure des améliorations de textures des murs pour renforcer l'aspect visuel, ou même des révisions des portes de sortie pour les rendre plus cohérentes avec le reste du design de la prison.

En résumé, notre prochaine étape consiste à peaufiner le décor intérieur de la prison pour qu'il corresponde parfaitement au scénario et à l'ambiance recherchée, tout en restant attentifs aux éventuelles améliorations à apporter à l'extérieur.

4.1.2 Personnage

Pour adapter le personnage principal au personnage souhaité lié au scénario, nous devons trouver et mettre en place un modèle 3D d'un prisonnier qui puisse être intégré à Unity et adapté à toutes les mécaniques du jeu. Nous veillerons également à optimiser le modèle pour garantir des performances fluides du jeu.

4.1.3 Garde

Nous prévoyons de modifier le design du garde de la prison pour qu'il corresponde à l'apparence du docteur présent dans le scénario. Notre prochaine étape consistera donc à trouver ou créer un modèle 3D du docteur et à l'intégrer dans Unity pour qu'il puisse parcourir la prison et interagir avec les joueurs selon les mécaniques de jeu établies.

4.1.4 Objets

L'import d'objets 3D essentiels, tels que des clés ou des boîtes, étaient manquants pour résoudre les énigmes du jeu. Ces objets joueront un rôle crucial dans l'avancement du joueur en lui permettant de surmonter les obstacles et de progresser dans l'histoire.

4.1.5 Menu du jeu

Nous projetons de concevoir le design du menu du jeu, une composante essentielle pour l'expérience de jeu. Nous avions déjà élaboré une page de couverture pour le jeu et nous avions l'intention d'adapter ce design pour le menu. Cette démarche a été facilitée par l'utilisation d'une intelligence artificielle. Cette couverture sera finalement abandonnée.

4.2 Mécanique de jeu

Pour ce qui est des mécaniques du jeu nous prévoyons d'ajouter des mouvements et des animations pour pouvoir se déplacer tout en étant accroupis, pouvoir accélérer lors du déplacement vers l'arrière et pouvoir sauter vers l'avant lorsque l'on sprint et saute simultanément.



FIGURE 32 – Couverture de la Grande Evasion

4.2.1 Inventaire

Nous incluons également la création d'un système d'inventaire dans nos objectifs pour finaliser le projet. Cet inventaire permettra aux joueurs de stocker les différents objets nécessaires pour résoudre les énigmes et s'évader de la prison. Nous avons l'intention de concevoir un inventaire capable de stocker jusqu'à cinq objets visibles en plein jeu, ainsi qu'un inventaire étendu pouvant être ouvert pour présenter plusieurs espaces disponibles pour différents objets. Les objets pourront être récupérés et placés directement dans l'inventaire, et les joueurs auront également la possibilité de jeter des éléments de leur inventaire selon les besoins de la situation.

4.3 Intelligence Artificielle (IA)

4.3.1 Mouvement IA

Il reste à ajouter une partie de code permettant au garde de patrouiller dans une partie de la salle, et de poursuivre le prisonnier uniquement s'il entre dans son champ de vision et/ou d'audition. De plus, un système de dégâts doit être intégré au code afin que le garde puisse renvoyer le prisonnier dans sa cellule.

4.3.2 Animation IA

On ajoutera une animation qui fera en sorte que le garde ne soit pas en position bras tendu : nous ajouterons une animation lorsqu' il est immobile et lorsqu' il est en train de se déplacer.

4.4 Mode multijoueur

Le mode multijoueur est un élément essentiel de notre projet, et nous avons déjà entamé des recherches à ce sujet. Nous avons identifié Photon Engine comme un moteur de réseau en temps réel spécialement conçu pour les jeux multijoueurs. Ce moteur propose des fonctionnalités avancées telles que le matchmaking et la communication entre les joueurs. Ainsi, nous avons initié le processus d'intégration des fondations de ce moteur dans Unity pour développer notre mode multijoueur.

4.5 Site Web

Pour le développement du site Web, nous prévoyons d'intégrer des étapes supplémentaires au fil de l'avancement du jeu. Nous désactiverons temporairement les boutons de téléchargement qui ne sont pas encore opérationnels, mais qui seront rendus disponibles dès que les documents correspondants pourront être mis en ligne.

5 Avancement après la première soutenance technique

Durant la deuxième et dernière période de notre projet, nous avons implémenter des fonctionnalités déjà prévues mais nous avons aussi remis en cause certains choix pour améliorer l'expérience de jeu de l'utilisateur. Nous avons aussi totalement repensé nos logos d'entreprise et de jeu.

5.1 Design du jeu

5.1.1 Prison

Phase Suivante du Développement

Dans la phase suivante du développement de la prison, nos prévisions incluaient :

- **Décors Intérieurs** : Intégration d'équipements médicaux et d'autres détails pour enrichir l'immersion et fournir des indices pour les énigmes.
- **Réajustement** : Ouverture à des ajustements potentiels pour améliorer les textures des murs et aligner les portes de sortie avec le design global de la prison.

Améliorations Réalisées

Depuis les phases initiales du projet, les améliorations suivantes ont été réalisées :

- **Niveau Inférieur** : Les salles ont été spécifiquement aménagées pour refléter un cabinet médical, avec l'inclusion d'éléments tels que des lits, des armoires, des fioles de sang, des crânes et d'autres éléments, maintenant ainsi l'ambiance du docteur fou.
- **Éclairage** : Une ambiance sombre a été choisie avec des lumières clignotantes pour maintenir une tension constante et une immersion accrue.

-
- **Portes de la Prison :** Les portes ont été redessinées pour s'aligner avec l'atmosphère carcérale du jeu, contribuant ainsi à l'authenticité et à l'immersion globale de l'expérience.



FIGURE 33 – Améliorations de la prison

Ces ajouts ont été soigneusement conçus pour enrichir l'expérience de jeu en offrant un environnement visuel captivant et en soutenant les éléments narratifs et de gameplay du jeu. Les objectifs initiaux ont été pleinement atteints, créant un décor convaincant et cohérent qui répond aux exigences du scénario et offre une expérience immersive aux joueurs.

5.1.2 Personnage

Phase Suivante du Développement

Dans la phase suivante du développement de la prison, nos prévisions incluaient :

- **Personnages** : Adoption de tenues de détenus orange pour renforcer visuellement l'emprisonnement et l'immersion du jeu.



FIGURE 34 – Nouveau modèle de personnage

Améliorations Réalisées

Depuis les phases initiales du projet, les améliorations suivantes ont été réalisées :

- **Modélisation des Personnages** : Sélection et intégration d'un modèle 3D de personnage trouvé sur Sketchfab, adapté pour une utilisation aussi bien en mode solo qu'en mode multijoueur. Ce choix simplifie la gestion des ressources et garantit une cohérence visuelle à travers le jeu.
- **Animations et Interactions** : Configuration des animations de déplacement, d'interaction avec les objets, et d'actions spécifiques nécessaires pour résoudre les

énigmes et progresser dans le jeu.

- **Intégration dans l'environnement:** Harmonisation du modèle de personnage avec l'environnement immersif de la prison, créant ainsi une expérience de jeu cohérente et engageante où chaque élément visuel contribue à l'atmosphère générale et au récit du jeu.

Ces développements ont été soigneusement orchestrés pour enrichir l'expérience des joueurs en offrant des personnages visuellement distincts et fonctionnels, alignés avec le contexte narratif et les mécaniques de gameplay du jeu.

5.1.3 Garde

Améliorations Réalisées

Depuis les premières étapes du projet, plusieurs améliorations ont été apportées au personnage de « The Plague », remplaçant le rôle traditionnel du garde de la prison

Modèle Sélectionné

Un modèle de médecin de peste a été choisi sur l'Unity Asset Store pour son adéquation parfaite avec le thème sombre et oppressant du jeu, facilitant ainsi son intégration dans l'environnement.

Éléments Distinctifs Ajoutés

- **Lanterne :** « The Plague » est équipé d'une lanterne qu'il tient en main, non seulement pour éclairer son chemin à travers la prison sombre, mais aussi pour ajouter une touche de réalisme et de mystère à son personnage.
- **Yeux Rouges Lumineux :** Des yeux rouges ont été ajoutés à « The Plague », projetant un faisceau lumineux pour intensifier les sensations de stress et d'immersion chez les joueurs. Cette caractéristique renforce l'atmosphère terrifiante du personnage, contribuant ainsi à l'oppression ressentie dans la prison.



FIGURE 35 – The Plague

Ces choix de conception ont été soigneusement planifiés pour permettre à « The Plague » de jouer un rôle central en tant qu’antagoniste principal du jeu. Sa présence menaçante et ses capacités spécifiques, notamment sa lanterne et ses yeux lumineux, créent une tension palpable qui augmente le danger perçu par les joueurs et rend l’évasion encore plus captivante.

5.1.4 Objets

Améliorations Réalisées

Depuis les phases initiales du projet, les améliorations suivantes ont été réalisées concernant les objets interactifs :

- **Sélection et Placement :** Choix soigneux et placement stratégique d'objets tels que clés, calendriers, bloc-notes, tableaux et digicode, formant une base cruciale pour enrichir l'expérience de jeu et offrir des indices logiques intégrés à l'environnement de la prison.
- **Source des Objets :** Récupération principalement sur Sketchfab, une plateforme proposant une large variété de modèles 3D, facilitant leur importation et leur adaptation dans Unity pour une intégration fluide dans le jeu.

Ces ajouts ont été méticuleusement conçus pour soutenir les énigmes et les défis du jeu, offrant aux joueurs une expérience immersive où chaque objet interactif contribue de manière significative à l'avancement et à la résolution des puzzles, tout en maintenant la cohérence avec le scénario et l'ambiance générale du jeu.

5.1.5 Menu du jeu

Améliorations Réalisées

Depuis les phases initiales du développement du menu principal, les améliorations suivantes ont été réalisées :

- **Design Visuel :** Création d'un menu principalement dominé par les couleurs rouge et noir, symbolisant respectivement le danger, l'excitation et le côté sombre, renforçant ainsi l'ambiance du jeu.
- **Intégration d'Images :** Utilisation d'images trouvées sur le web, telles qu'une re-

présentation de la peste en noir et rouge, pour créer un style visuel distinctif et captivant.

- **Police de Caractères** : Choix de la police "Who Asks Satan", utilisée de manière répétée pour renforcer l'identité visuelle cohérente et distinctive du jeu.
- **Design de Boutons** : Création d'un design de bouton unique, utilisé à diverses fins dans le jeu pour améliorer l'expérience utilisateur.
- **Éléments Interactifs** : Ajout de fonctionnalités interactives telles que des boutons pour créer une partie et rejoindre une partie, accompagnés de champs pour saisir un nom de serveur et un pseudonyme de joueur.
- **Animation et Son** : Intégration d'animations de zoom sur les boutons lors du survol de la souris, et d'effets sonores de clic pour renforcer l'interactivité et l'immersion.



FIGURE 36 – Menu du jeu

Ces choix de conception et d'interaction contribuent à une première impression immersive et engageante, alignant parfaitement le menu principal avec l'ambiance générale du jeu et offrant une expérience utilisateur captivante dès le début.

5.1.6 Option

Comme tout jeu, un menu d'options est nécessaire pour offrir à chaque joueur une expérience personnalisée. Nous avons décidé de créer un menu d'options tout en conservant le même design et style que le menu principal du jeu. Ce menu d'options, accessible en cours de partie, comporte trois sliders permettant d'ajuster différents paramètres :

- **La sensibilité** : Le slider de sensibilité permet aux joueurs d'ajuster la réactivité des contrôles pour une expérience de jeu optimale.
- **La musique** : Ce slider permet de régler le volume de la musique d'ambiance du jeu, offrant ainsi une immersion sonore personnalisée.
- **Les SFX (effets sonores)** : Ce slider permet de régler le volume des effets sonores, tels que les bruits de pas ou la respiration, pour enrichir l'expérience audio du joueur.

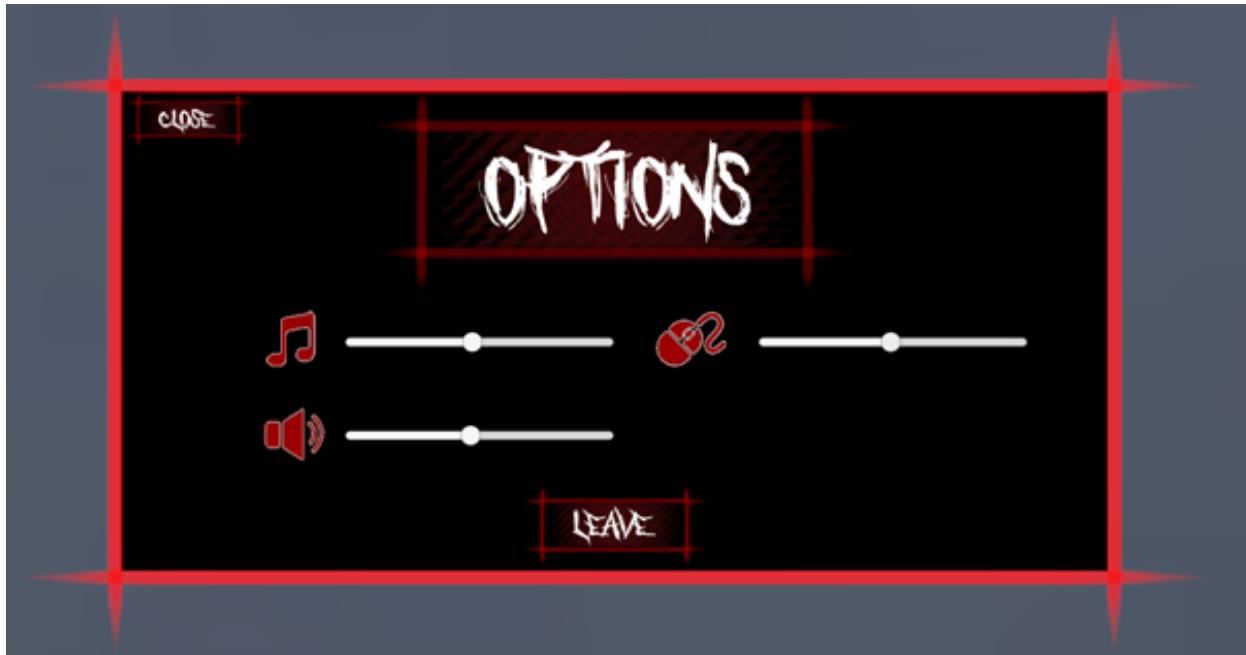


FIGURE 37 – Menu options

Ce menu d'options est essentiel pour garantir que chaque joueur puisse adapter le jeu à ses besoins spécifiques, assurant ainsi une expérience de jeu confortable et immersive.

5.1.7 Interfaces de victoire et de défaite

Dans « La Grande Évasion », les joueurs peuvent gagner ensemble en s'échappant de la prison, mais ils peuvent aussi perdre si l'un des joueurs meurt. Pour marquer ces moments, nous avons conçu des écrans de fin de partie fidèles au style visuel du jeu.

Lorsque les joueurs gagnent, un écran de victoire apparaît, célébrant leur évasion réussie. Cet écran utilise les mêmes éléments de design que le reste du jeu pour maintenir une cohérence visuelle.



FIGURE 38 – Écran de victoire

En cas de défaite, un interface similaire informe les joueurs de leur échec. Ce design, tout en étant distinct, conserve l'identité visuelle du jeu pour assurer une expérience cohérente jusqu'à la fin.



FIGURE 39 – Écran de défaite

Ces écrans de fin de partie sont essentiels pour donner aux joueurs un retour clair et immersif sur leur performance, renforçant ainsi l'expérience globale du jeu.

5.2 Mécaniques du jeu

5.2.1 Mouvements du joueur

Les déplacements du joueur sont un élément central de notre jeu, influençant directement l'expérience et l'immersion. C'est pourquoi nous les avons soigneusement retravaillés pour les rendre aussi fluides et agréables que possible.

- **Révision du script :** Nous avons repris le script depuis le début afin de résoudre divers problèmes rencontrés. Par exemple, nous avons ajouté une condition aux mouvements simples (avant, arrière, gauche, droite) car le joueur subissait des forces résiduelles indésirables sur son Rigidbody. Si le joueur ne bouge plus, les animations sont réinitialisées et les forces sur le Rigidbody sont remises à zéro (voir Figure 40).

```
if (!isMoving)
{
    StopAnimation();
    rb.velocity = Vector3.zero;
    rb.angularVelocity = Vector3.zero;
}
```

FIGURE 40 – Animations réinitialisées.

- **Utilisation de `transform.Translate`** : Désormais, les fonctions de déplacement utilisent `transform.Translate` au lieu de `rb.MovePosition`.
- **Détection de chemin libre** : Nous avons créé un booléen pour déterminer si le chemin dans la direction où le joueur regarde est libre, afin d'éliminer les bugs de collision (voir Figure 41). La fonction `IsPathClear` vérifie également si l'élément rencontré n'a pas le tag `ThroughObject`. Cela permet de garder certaines collisions, notamment celles sur les escaliers ou sur les `Box Collider` de certains objets pour permettre au joueur d'interagir avec eux.

```
public bool IsPathClear(Vector3 direction, float moveDistance)
{
    float rayDistance = moveDistance + 0.2f; // Ajustez cette distance selon vos besoins
    float rayHeightOffset = 1.0f; // Décalage de hauteur depuis le sol

    // Calculer le point de départ du rayon à une hauteur spécifique
    Vector3 rayStart = transform.position + Vector3.up * rayHeightOffset;

    RaycastHit hit;

    // Lance le rayon et récupère l'information de collision
    bool isHit = Physics.Raycast(rayStart, direction, out hit, rayDistance);

    // Dessine une ligne pour visualiser le trajet du rayon
    Debug.DrawLine(rayStart, rayStart + direction * rayDistance, isHit ? Color.red : Color.green);

    if (isHit)
    {
        if (hit.collider != null && !hit.collider.CompareTag("ThroughObject"))
        {
            return false; // Il y a un obstacle qui n'est pas un objet de traversée
        }
    }
    return true; // Pas d'obstacle
```

FIGURE 41 – Vérification de la clarté du chemin.

- **Simplification des mécanismes de déplacement** : Nous avons essayé d'ajouter les fonctionnalités prévues, telles que le saut vers l'avant, l'accroupissement et le sprint arrière. Cependant, nous nous sommes rapidement aperçus que ces fonctionnalités étaient inutiles pour le bon déroulement du jeu. Non seulement elles surchargeaient le joueur d'informations pendant le didacticiel, mais elles créaient également des bugs qui nuisent à l'expérience globale. Après plusieurs tests et retours, il est devenu évident que simplifier les mécanismes de déplacement était la solution optimale, sans compromettre le plaisir du jeu.

5.2.2 Animations du joueur

Comme il y avait des bugs dans les mouvements, il y en avait aussi pour les animations. L'animation du joueur se réinitialisait toute seule lors des déplacements.

- **Activation spécifique des animations :** Nous avons veillé à activer chaque animation spécifiquement lors de son mouvement associé et à les désactiver pour tout autre mouvement.
- **Fonction StopAnimation :** Pour nous faciliter la tâche, nous avons créé une fonction StopAnimation qui arrête toutes les animations (voir Figure 46).

```
void StopAnimation()
{
    // Désactiver toutes les animations de marche
    PlayerAnimator.SetBool("run", false);
    PlayerAnimator.SetBool("walk", false);
    PlayerAnimator.SetBool("walk left", false);
    PlayerAnimator.SetBool("walk right", false);
    PlayerAnimator.SetBool("walk back", false);
}
```

FIGURE 42 – Arrêt des animations.

5.2.3 Inventaire

Pour la réalisation de notre jeu, un système d'inventaire est nécessaire. Nous avons donc repris le design du menu pour concevoir l'inventaire, permettant ainsi de conserver une identité propre à « La Grande Évasion ».

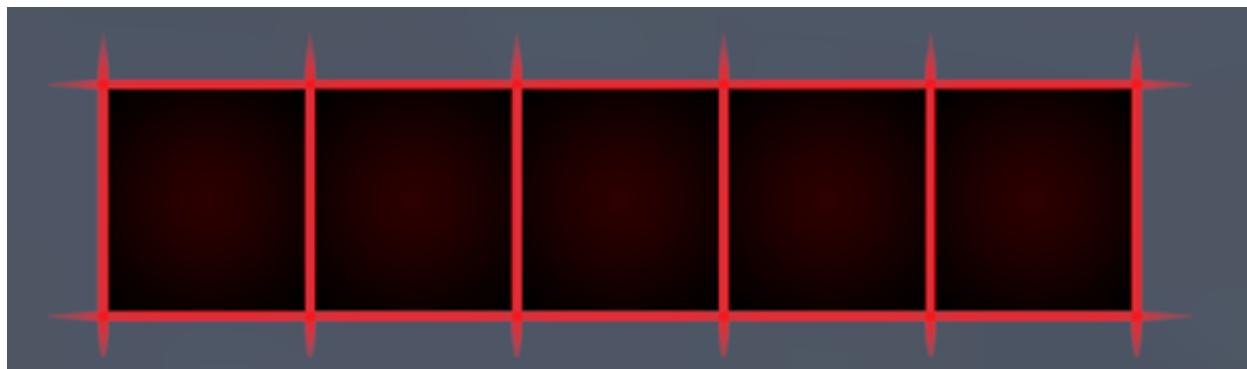


FIGURE 43 – Inventaire

Chaque objet pouvant être ramassé est associé à un layer spécifique et à un script nommé "Item". Ce script comprend une variable publique de type Sprite nommée icon, destinée à conserver l'image associée à l'objet. Le script "Item" inclut également une méthode DeactivateItem qui permet de désactiver l'objet.

```
public class Item : MonoBehaviourPun
{
    0 references
    public Sprite icon; // Conserve l'image de l'item en question

    [PunRPC]
    0 references
    public void DeactivateItem()
    {
        gameObject.SetActive(false);
    } // désactive l'item pour tous les joueurs en multi
}
```

FIGURE 44 – Item

Le script "Inventaire", attaché au personnage contrôlé par le joueur, gère l'interaction avec les objets et leur affichage dans les slots de l'inventaire. Dans la méthode Update, il vérifie si la touche d'interaction est pressée et s'assure que l'interaction s'applique uniquement pour le joueur local en appelant la méthode Interact.

```
private void Update()
{
    if (photonView.IsMine && Input.GetKeyDown(KeyCode.E)) // l'interaction se fait seulement pour le joueur local
    {
        Interact();
    }
}
```

FIGURE 45 – Interaction

Dans la méthode Interact, un raycast est utilisé pour détecter les objets de type "Item" en collision avec le collider de l'item ayant le layer approprié. Si un objet est détecté, la fonction AddItemToInventory est appelée sur tous les clients, en passant l'ID de vue de l'objet. La méthode AddItemToInventory récupère le composant Item attaché au PhotonView de l'objet et cherche un slot libre dans l'inventaire. Une fois ce slot trouvé, elle ajoute l'item à ce slot et désactive l'item pour tous les joueurs en multijoueur, garantissant une synchronisation cohérente des objets dans le jeu.

```
[PunRPC]
0 references
private void AddItemToInventory(int itemViewID)
{
    PhotonView itemPhotonView = PhotonView.Find(itemViewID);
    if (itemPhotonView != null)
    {
        Item itemToAdd = itemPhotonView.GetComponent<Item>(); // récupère le composant Item
        Slot openSlot = null; // initialise un slot
        foreach (Slot slot in InventorySlots) // parcourt les slots pour chercher un emplacement vide
        {
            if (slot.GetItem() == null)
            {
                openSlot = slot; // définit le slot libre à utiliser
                break;
            }
        }
        if (openSlot != null)
        {
            openSlot.SetItem(itemToAdd); // ajoute l'item au slot
            itemPhotonView.RPC("DeactivateItem", RpcTarget.AllBuffered); // désactive l'item en multi
        }
    }
}
```

FIGURE 46 – Ajouter un item à l'inventaire

Ce système d'inventaire assure une gestion fluide et immersive des objets ramassés par les joueurs, tout en maintenant une cohérence visuelle et fonctionnelle avec le reste du jeu.

5.2.4 Énigmes

Les énigmes sont essentielles car elles forment la base de notre jeu, et leur résolution est le seul moyen de s'échapper. Nous avons créé différentes énigmes de niveaux de difficulté variés, allant de simples recherches à des réflexions plus complexes.

Énigme de la Cellule

Pour résoudre cette énigme individuelle, les joueurs doivent localiser les clés nécessaires pour ouvrir leur cellule. Un GameObject avec un Collider "Is Trigger" détecte l'interaction du joueur, déclenchant une séquence pour vérifier si le joueur possède la clé requise.

```
private void Update()
{
    if (isPlayerInTrigger && Input.GetKeyDown(KeyCode.E))
    {
        TryToOpenDoor();
    }
}
```

FIGURE 47 – Conditions d'ouverture de la porte

Le script utilise une fonction TryToOpenDoor qui vérifie si la porte est verrouillée et si le joueur possède la clé adéquate, facilitant ainsi l'évasion individuelle.

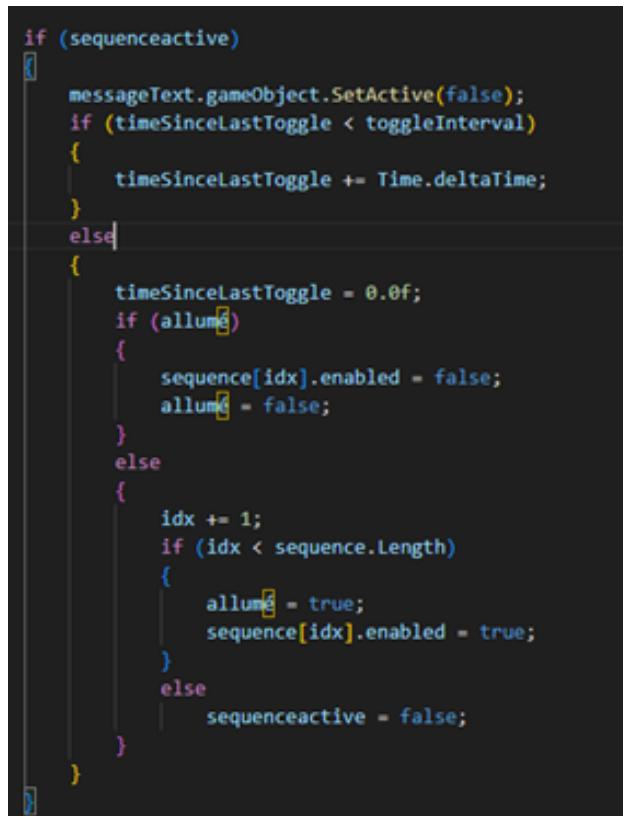
```
private void TryToOpenDoor()
{
    if (!isLocked || CheckForKey(requiredKeySprite))
    {
        myDoor.Play("Opendoor2", 0, 0.0f);
        isLocked = false;
        StartCoroutine(OpenDoorSequence());
    }
    else
    {
        StartCoroutine(DisplayMessage("You need a key to open this door!", 2));
    }
}
```

FIGURE 48 – Vérification pour ouvrir la porte

Énigmes de la Salle Principale

Après avoir résolu l'éénigme de la cellule, les joueurs doivent coopérer pour résoudre plusieurs défis dans la salle principale, nécessitant une combinaison de compétences individuelles.

- **Énigme Rouge :** Utilisation d'un calendrier et de notes pour décoder une séquence visuelle cruciale, sans nécessiter de code.
- **Énigme Bleue :** Résolution d'un problème mathématique complexe en utilisant des lumières colorées pour déterminer les valeurs de variables clés, associées à la couleur bleue. Le script vérifie si la séquence de lumières est active avec un booléen `sequenceactive`, ajustant les lumières en fonction des interactions du joueur pour résoudre le puzzle.



```
if (sequenceactive)
{
    messageText.gameObject.SetActive(false);
    if (timeSinceLastToggle < toggleInterval)
    {
        timeSinceLastToggle += Time.deltaTime;
    }
    else
    {
        timeSinceLastToggle = 0.0f;
        if (allumé)
        {
            sequence[idx].enabled = false;
            allumé = false;
        }
        else
        {
            idx += 1;
            if (idx < sequence.Length)
            {
                allumé = true;
                sequence[idx].enabled = true;
            }
            else
                sequenceactive = false;
        }
    }
}
```

FIGURE 49 – Vérification séquence de lumières

-
- **Énigme Verte** : Alignement précis des tableaux pour révéler un chiffre vert essentiel, nécessitant une interaction directe du joueur pour tourner les tableaux dans la bonne direction. Le script détecte la présence du joueur via `OnTriggerEnter` et active la rotation des tableaux lorsque des touches spécifiques sont pressées (G, H, J).

```
if (angle1 == 180 && angle2 == 225 && angle3 == 0)
{
    num.gameObject.SetActive(true);
}
else
{
    num.gameObject.SetActive(false);
    if (playerInZone && photonViewplayer.IsMine)
    {
        if (Input.GetKeyDown(KeyCode.G))
        {
            RotateTableau(tableau1);
        }
        if (Input.GetKeyDown(KeyCode.H))
        {
            RotateTableau(tableau2);
        }
        if (Input.GetKeyDown(KeyCode.J))
        {
            RotateTableau(tableau3);
        }
    }
}
```

FIGURE 50 – Rotation des tableaux

- **Énigme Jaune** : Superposition d’éléments pour déchiffrer une séquence de lettres cachée, utilisée pour résoudre une énigme de code ultérieure.

Digicode

Utilisation d'un digicode final reliant les informations collectées des énigmes précédentes (couleurs et chiffres) pour ouvrir la voie vers la liberté.

5.2.5 Vies

Nous avons introduit une barre de vie pour tous les personnages. Lorsqu'un joueur est touché par le docteur, il est renvoyé à son point de spawn et perd un quart de sa barre de vie. Si un joueur atteint zéro points de vie, la partie se termine pour tous les joueurs, car tous les prisonniers doivent demeurer en vie pour réussir leur évasion.

```
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Agent") && !isFrozen)
    {
        StartCoroutine(FreezeAndTeleport());
        TakeDamage(25f);
    }
}
```

FIGURE 51 – Détecteur de collision

Si un joueur entre en collision avec le docteur (dont le tag est "Agent"), il se téléporte instantanément à son point de spawn à l'aide de la fonction "FreezeAndTeleport" et subit 25 points de dégâts avec la fonction "TakeDamage".

```
public void TakeDamage(float damage)
{
    currentHealth -= damage;
    if (currentHealth <= 0)
    {
        currentHealth = 0;
        photonView.RPC("ActivateCanvasAndDisableAudioForAll2", RpcTarget.All);
    }
    UpdateHealthBar();
}
```

FIGURE 52 – Inflige les dégâts

Enfin, dans la fonction "TakeDamage", nous avons inclus une vérification afin que si l'un des joueurs atteint 0 point de vie, tous les joueurs voient l'écran de défaite s'afficher. Pour cela, nous utilisons la commande : `photonView.RPC("ActivateCanvasAndDisableAudioForAll2", RpcTarget.All);`

5.2.6 Sons

Pour garantir une immersion optimale, nous avons intégré divers sons essentiels à l'expérience du jeu :

- **Musique d'Ambiance** : Une musique d'ambiance enveloppante est jouée en continu, créant une atmosphère constante quel que soit l'endroit où se trouve le joueur.
- **Sons de Pas** : Chaque joueur entend le son de ses propres pas uniquement lorsqu'il est en mouvement. Le script vérifie si le joueur se déplace et si le chemin devant lui est libre, déclenchant les bruits de pas toutes les 0.5 secondes en marchant et toutes les 0.3 secondes en courant.

```
void HandleFootsteps(Vector3 dir)
{
    if (IsMoving() && IsPathClear(dir, 1))
    {
        footstepTimer -= Time.deltaTime;
        if (footstepTimer <= 0f)
        {
            if (footstep AudioSource != null)
            {
                footstep AudioSource.Play();
            }
            footstepTimer = IsRunning() ? runningFootstepInterval : footstepInterval;
        }
        else
        {
            footstepTimer = footstepInterval;
        }

        lastPosition = transform.position;
    }
}
```

FIGURE 53 – Bruits de pas

- **Son de Respiration** : Un son de respiration est présent en permanence, devenant plus intense lorsque le joueur court, intensifiant ainsi l'immersion.

```
void HandleBreathing()
{
    if (breathing AudioSource != null)
    {
        // Ajuster la fréquence (pitch) de la respiration en fonction de l'état de course
        breathing AudioSource.pitch = IsRunning() ? runningBreathingPitch : normalBreathingPitch;
    }
}
```

FIGURE 54 – Son de la respiration

- **Battement de Cœur :** L'utilisateur entend un battement de cœur seulement lorsque le docteur le poursuit, ajoutant une tension dramatique lors des poursuites.

```
if (CanSeePlayer())
{
    StartChase();
    Heart.Play();
}
```

FIGURE 55 – Battement de coeur

- **Dialogues du Docteur :** À intervalles réguliers de trente secondes, le docteur énonce la phrase menaçante : "I will kill you", renforçant ainsi la présence opprimeante du personnage antagoniste.

```
IEnumerator PlayAudioPeriodically()
{
    while (true)
    {
        yield return new WaitForSeconds(30f);
        PlayAudio();
    }
}
```

FIGURE 56 – Dialogue de The Plague

5.3 Intelligence Artificielle

5.3.1 Mouvements

Tous les objectifs établis ont été entièrement atteints. En outre, un champ de vision et des éléments sonores visibles ont été intégrés pour enrichir l'expérience utilisateur avec The Plague.

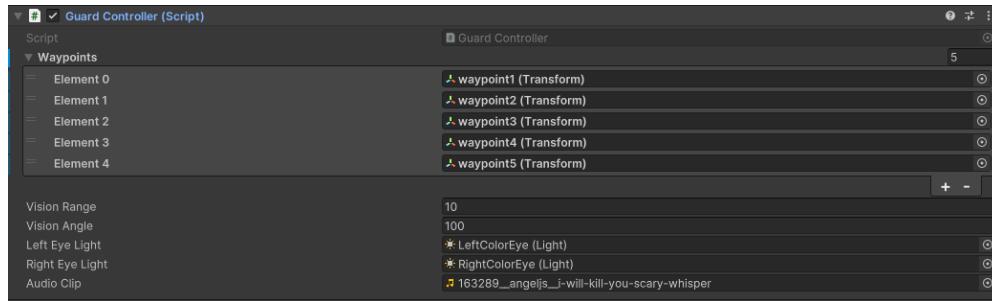


FIGURE 57 – Composant Guard Controller

Le nouveau script ajoute plusieurs fonctionnalités essentielles et sophistiquées, augmentant ainsi les capacités de l'IA du garde.

Au démarrage (`Start()`), les composants critiques comme `navMeshAgent` pour la navigation et `animator` pour les animations sont initialisés. Cela assure que le garde est prêt à interagir dynamiquement avec l'environnement du jeu dès le début.

```
void Start()
{
    navMeshAgent = GetComponent<NavMeshAgent>();
    animator = GetComponent<Animator>();
    audioSource = GetComponent< AudioSource >();

    // Ajustez les paramètres de vitesse ici
    navMeshAgent.speed = 3.5f;
    navMeshAgent.acceleration = 8f;
    navMeshAgent.angularSpeed = 120f;

    StartCoroutine(WaitAndFindPrisoner());
    StartCoroutine(PlayAudioPeriodically());
}
```

FIGURE 58 – Declaration et initialisation des composants

Ce script utilise une énumération (`GuardState`) pour définir les différents états du garde. La méthode `Update()` gère la logique de changement d'état en fonction de la valeur de `currentState`. Les animations sont mises à jour en fonction de la vitesse et de la direction du `navMeshAgent`, ce qui rend le comportement du garde plus naturel et réactif.

L'IA du garde est structurée pour passer fluidement entre différents états (`GuardState`), tels que la patrouille et la poursuite. Cette modularité permet une réponse rapide aux changements de contexte, améliorant ainsi l'immersion et l'interaction joueur-IA.

```
private enum GuardState
{
    Patrol,
    Chase
}

private GuardState currentState;

void Update()
{
    switch (currentState)
    {
        case GuardState.Patrol:
            UpdatePatrol();
            break;
        case GuardState.Chase:
            UpdateChase();
            break;
        default:
            break;
    }

    // Mettre à jour les paramètres d'animation en fonction de la vitesse et de la direction du NavMeshAgent
    Vector3 localVelocity = transform.InverseTransformDirection(navMeshAgent.velocity);
    animator.SetFloat("Speed", localVelocity.z);
    animator.SetFloat("Direction", localVelocity.x);
}
```

FIGURE 59 – Déclaration des variables et champs du garde

Ce script utilise une énumération (`GuardState`) pour définir les différents états du garde. La méthode `Update()` gère la logique de changement d'état en fonction de la valeur de `currentState`. Les animations sont mises à jour en fonction de la vitesse et de la direction du `navMeshAgent`, ce qui rend le comportement du garde plus naturel et réactif.

À l'aide des méthodes `FindPrisoner()` et `CanSeePlayer()`, le garde est programmé pour détecter activement la présence du joueur dans son environnement. Cette capacité de détection est essentielle pour ajuster son comportement de manière appropriée, en passant de la patrouille à la poursuite lorsque nécessaire.

```
public bool CanSeePlayer()
{
    if (player == null)
    {
        return false;
    }

    Vector3 direction = (player.position - transform.position).normalized;
    float angle = Vector3.Angle(transform.forward, direction);

    if (angle < visionAngle / 2)
    {
        RaycastHit hit;
        if (Physics.Raycast(transform.position, direction, out hit, visionRange))
        {
            if (hit.collider.CompareTag("Player"))
            {
                return true;
            }
        }
    }

    return false;
}

public void FindPrisoner()
{
    GameObject prisoner = GameObject.FindGameObjectWithTag("Player");
    if (prisoner != null)
    {
        player = prisoner.transform;
    }
    else
    {
        Debug.LogError("Prisoner not found! Make sure the prisoner has the 'Player' tag.");
    }
}
```

FIGURE 60 – Détection et Interaction avec le Joueur

Ces méthodes permettent au garde de détecter la présence du joueur. `CanSeePlayer()` contient la logique de détection visuelle, tandis que `FindPrisoner()` définit comment le garde localise le joueur dans la scène. Ces fonctions sont cruciales pour que le garde passe de l'état de patrouille à l'état de poursuite.

Pendant la patrouille (`UpdatePatrol()`), le garde utilise des waypoints pour naviguer de manière autonome à travers la scène. Lorsque le joueur est détecté, il passe à l'état de poursuite (`UpdateChase()`), ajustant sa trajectoire pour suivre activement le joueur, augmentant ainsi le défi et l'engagement du joueur.

```
void UpdatePatrol()
{
    if (waypoints.Length == 0)
    {
        return;
    }

    if (!navMeshAgent.pathPending && (navMeshAgent.remainingDistance <= navMeshAgent.stoppingDistance || navMeshAgent.velocity.sqrMagnitude == 0f))
    {
        if (!isWaitingAtWaypoint)
        {
            StartCoroutine(WaitAtWaypoint());
        }
    }

    // Vérifier si le joueur est visible pour passer à l'état Chase
    if (CanSeePlayer())
    {
        StartChase();
    }
}

void UpdateChase()
{
    // Continuer de suivre le joueur
    ChasePlayer();

    // Vérifier si le joueur n'est plus visible pour revenir à l'état Patrol
    if (!CanSeePlayer())
    {
        currentState = GuardState.Patrol;
    }
}
```

Act
Acc

FIGURE 61 – Navigation Autonome et Poursuite Dynamique

`UpdatePatrol()` gère la navigation du garde entre différents points de la scène (waypoints). `UpdateChase()` modifie la logique de déplacement pour suivre le joueur lorsqu'il est détecté. Cette transition entre la patrouille et la poursuite rend le comportement du garde plus dynamique et engageant.

Le jeu périodique d'un clip audio à intervalles réguliers (`PlayAudioPeriodically()`) enrichit l'expérience du joueur en ajoutant une dimension sonore réaliste. Cela peut être utilisé pour signaler des événements critiques comme la détection du joueur par le garde, contribuant ainsi à une immersion accrue dans le jeu.

```
IEnumerator PlayAudioPeriodically()
{
    while (true)
    {
        yield return new WaitForSeconds(30f);
        PlayAudio();
    }
}

void PlayAudio()
{
    if (audioClip != null && AudioSource != null)
    {
        AudioSource.clip = audioClip;
        AudioSource.Play();
    }
}
```

FIGURE 62 – Audio Périodique pour l'Immersion

`PlayAudioPeriodically()` est une coroutine qui joue un clip audio à intervalles réguliers, améliorant ainsi l'immersion sonore du jeu. Ce son peut servir d'indicateur pour le joueur, signalant des moments critiques comme la détection du joueur par le garde.

5.3.2 Animations

Les travaux réalisés ont été en accord avec les prévisions post-soutenance. Une nouvelle animation a été intégrée pour animer le garde de manière appropriée selon ses états d'activité :

- Nous avons développé un Blend Tree 2D simple directionnel qui connecte les animations "standing torch idle", "standing torch walk", "standing torch walk right", "standing torch walk left" et "standing torch walk back". Ce blend tree utilise les paramètres "speed" pour moduler la vitesse du déplacement du garde et "direction" pour spécifier la direction dans laquelle il se déplace.
- Cette approche nous permet de créer des animations fluides et réactives qui améliorent la qualité visuelle et l'authenticité du comportement du garde dans le jeu.

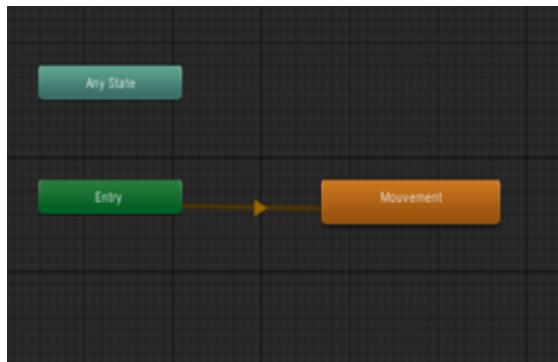


FIGURE 63 – Blend tree en tant que Layer default state

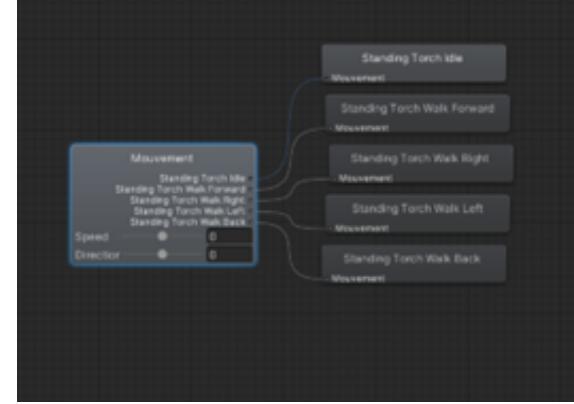


FIGURE 64 – Blend tree

Nous avons également dû ajouter les motions spécifiques pour chaque animation et ajuster les paramètres du Blend Tree afin d'assurer des transitions harmonieuses entre les différentes animations en fonction de la vitesse et de la direction du garde.

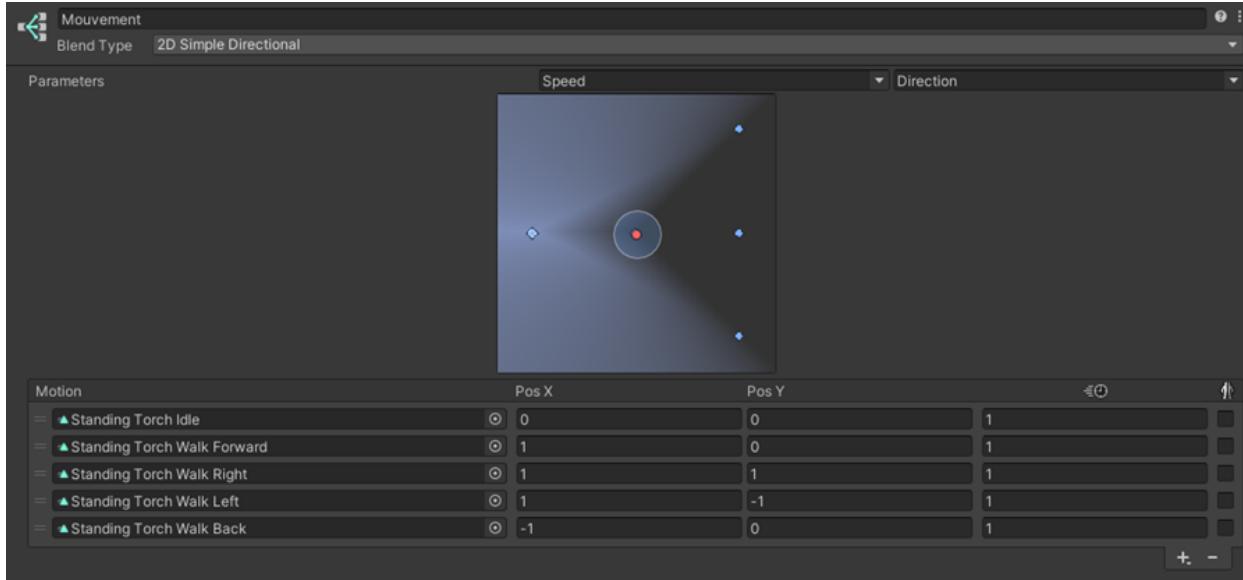


FIGURE 65 – Parametre blend tree

5.3.3 Représentation du champ de vision et du son

Nous avons intégré des yeux lumineux émettant une lumière rouge pour représenter le champ de vision du garde. Lorsqu'une personne pénètre dans ce champ de vision, notre système déclenche la poursuite par le garde, conformément au fonctionnement du code existant.

```
[SerializeField] private float visionRange = 10f;  
[SerializeField] private float visionAngle = 100f;  
[SerializeField] private Light leftEyeLight;  
[SerializeField] private Light rightEyeLight;
```

FIGURE 66 – Initialisation du Champ de Vision et Effets Sonores

En outre, dès qu'un prisonnier est enveloppé par cette lumière rouge, nous activons des battements de cœur audibles, accompagnés d'une respiration de plus en plus soutenue pour le prisonnier. De plus, le garde énonce toutes les 30 secondes : "Je vais te tuer".



FIGURE 67 – Prisonnier dans le champ de vision

5.4 Mode Multijoueur

L'implémentation du mode multijoueur est un élément crucial de notre jeu, étant donné que ce dernier repose sur la coopération. Quoi de mieux que de pouvoir jouer avec des amis ?

Nous avons commencé par importer le package PUN 2 depuis l'Asset Store de Unity pour implémenter le multijoueur grâce à Photon Engine. Ensuite, nous nous sommes rendus sur le site photonengine.com où nous avons créé une application pour obtenir un App ID, indispensable pour l'identification de notre projet dans Unity.

Après avoir ajouté cette App ID dans l'onglet Setup Project de notre jeu, nous avons réadapté tous les scripts et ajouté divers éléments pour assurer le bon fonctionnement de notre jeu en mode multijoueur.

Ajout de Composants

Nous avons ajouté des Photon View à tous les éléments que nous souhaitions synchroniser en ligne, tels que :

- Les joueurs
- Le Docteur
- Les éléments interactifs (portes, tableaux, lumières, etc.)

Nous avons également ajouté des Photon Transform View et des Photon Animator View permettant respectivement de synchroniser les positions et les rotations, et les animations.

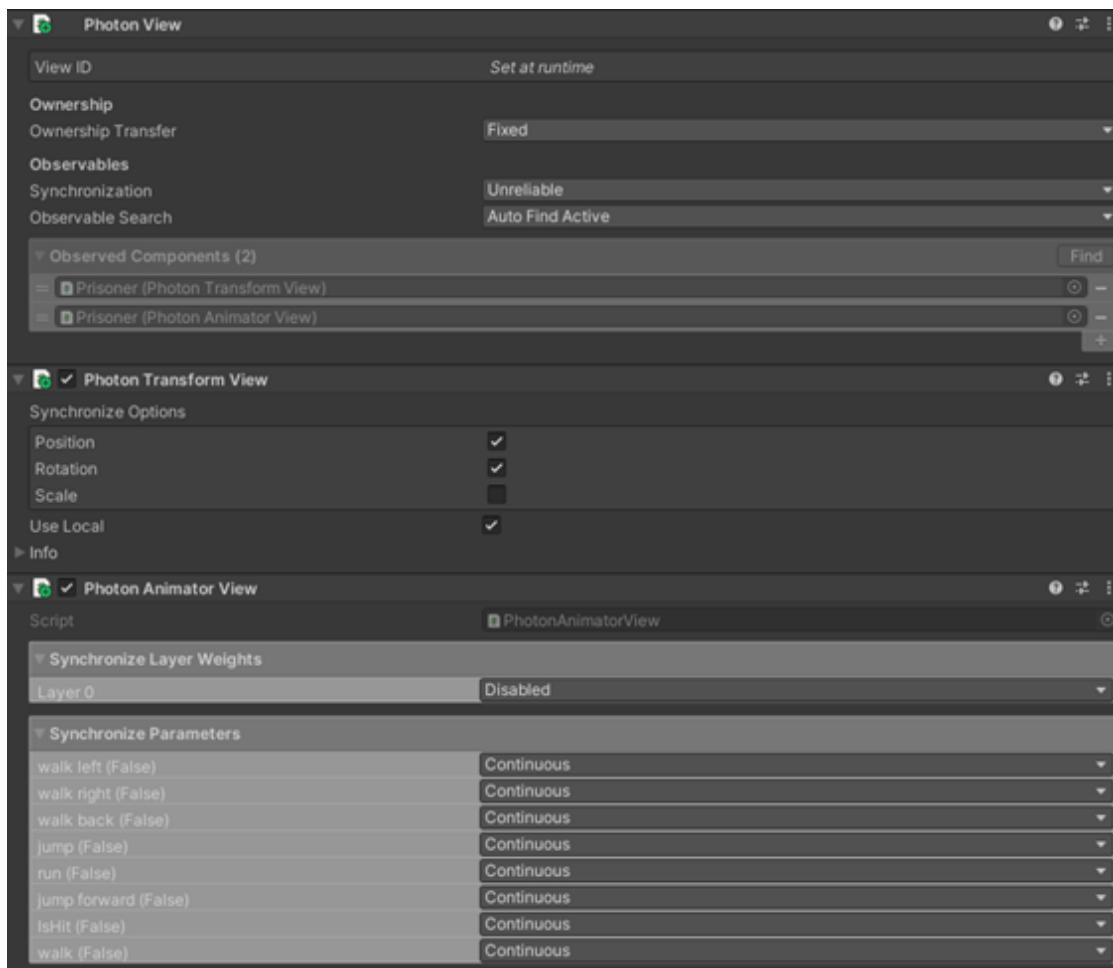


FIGURE 68 – Composants Photon

Adaptation des Scripts

Les scripts ont également dû être réadaptés. L'import de la bibliothèque Photon.Pun et l'héritage de MonoBehaviourPunCallbacks par le script sont obligatoires pour tous les scripts contenant une synchronisation en ligne :

```
using Photon.Pun;  
  
0 references  
public class Move : MonoBehaviourPunCallbacks
```

FIGURE 69 – Import de la bibliothèque Photon.Pun et Héritage

L'utilisation de la méthode IsMine sur les composants Photon View a été un des éléments essentiels pour l'adaptation du multijoueur. En effet, elle permet de savoir si le composant PhotonView (et donc l'objet auquel il est attaché) est sous le contrôle de l'instance locale du jeu. En d'autres termes, elle indique si l'objet est contrôlé par le client local.

```
if (photonView.IsMine && !OptionsMenuManager.OptionsMenuActive && !DigicodeTrigger.DigicodeActive)  
{  
    ProcessInput();
```

FIGURE 70 – Utilisation de IsMine pour le Contrôle Local

Ici, la fonction ProcessInput est celle qui permet au personnage de se déplacer, sauf qu'elle est active uniquement si le menu des options n'est pas activé, si le digicode n'est pas activé, et si le personnage est contrôlé par le client local. Cela permet à chaque joueur de contrôler son propre personnage et pas celui des autres.

L'utilisation des RPC (Remote Procedure Calls) a également été très utile. Elles nous permettent d'appeler des fonctions spécifiques sur des objets réseau de manière synchrone ou asynchrone, afin que tous les clients (ou un sous-ensemble spécifique de clients) exécutent la même fonction en même temps.

```
[PunRPC]
0 références
public void TakeDamage(float damage)
{
    currentHealth -= damage;
    if (currentHealth <= 0)
    {
        currentHealth = 0;
        photonView.RPC("ActivateCanvasAndDisableAudioForAll2", RpcTarget.All);
    }
    UpdateHealthBar();
}

[PunRPC]
0 références
public void ActivateCanvasAndDisableAudioForAll2()
{
    targetedCanvas.gameObject.SetActive(true);
    Cursor.visible = true;
    Cursor.lockState = CursorLockMode.None;

    // Désactiver tous les sons
    DisableAllAudioSources();
}
```

FIGURE 71 – Utilisation des RPC pour la Synchronisation

Dans cet exemple, si l'un des joueurs voit ses points de vie tomber à zéro, la ligne `photonView.RPC("ActivateCanvasAndDisableAudioForAll", RpcTarget.All);` s'active, ce qui déclenche la fonction `ActivateCanvasAndDisableAudioForAll` pour tous les joueurs (`RpcTarget.All`), affichant ainsi le canvas de défaite et désactivant les sources audio.

L'indication “[PunRPC]” au-dessus des fonctions est également nécessaire pour préciser que la fonction peut être appelée à distance sur d'autres clients connectés au même jeu.

5.5 Modification des logos

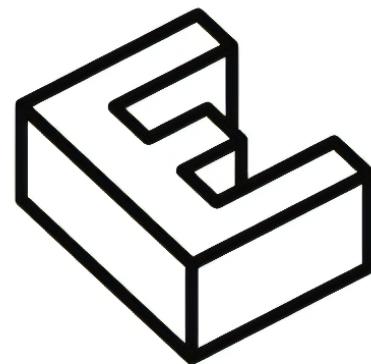
5.5.1 Comparaison ancien et nouveau logo

Logo d'entreprise

Pour ce qui est du côté marketing et image de marque de notre entreprise, nous avons décidé de changer entièrement notre logo (voir Figure ??). Le logo initial, généré par une intelligence artificielle, était trop basique, représentait mal notre entreprise et ne disait finalement rien sur l'identité de celle-ci. Il reprenait simplement notre nom.



(a) ancien logo



EPI GAMES

(b) nouveau logo

FIGURE 72 – Comparaison de l'ancien et du nouveau logo

Le nouveau logo ci-dessus est une création personnelle, donc naturellement plus à même de nous représenter. Le logo présente une lettre "E" en perspective, ce qui évoque immédiatement l'idée de la 3D, essentielle pour nos jeux d'escape game.

- **Simplicité et Distinction :** Le logo est simple mais distinctif.

- **Reconnaissabilité :** Il est donc plus facilement reconnaissable. Les joueurs se souviendront de notre entreprise.

Logo du Jeu

De la même manière, l'ancien logo du jeu était trop généraliste. Nous avons recréé un logo qui est plus propre à l'image de notre jeu.



(a) ancien logo



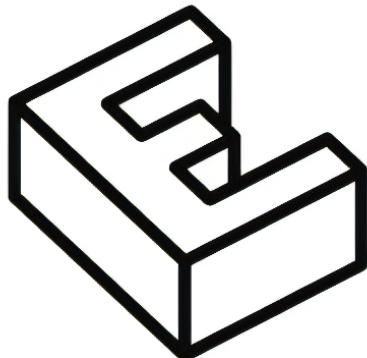
(b) nouveau logo

FIGURE 73 – Comparaison de l'ancien et du nouveau logo

5.5.2 Polyvalence du nouveau logo

Logo d'entreprise

Nous avons aussi cherché un logo polyvalent d'où le choix de couleurs simples avec lesquelles le logo peut garder sa cohérence en inversant les couleurs. On peut choisir le logo de gauche pour les surfaces claires et le logo de droite pour les surfaces sombres.



EPI GAMES

FIGURE 74 – Logo sur surface claire



EPI GAMES

FIGURE 75 – Logo sur surface sombre

Logo du Jeu

Nous voyons aussi ici que ce logo est plus polyvalent par rapport à la grande diversité de surfaces où un logo peut être apposé.



FIGURE 76 – Logo sur surface claire



FIGURE 77 – Logo sur surface sombre

5.6 Site Web

Pour le site web, nous avons décidé de modifier l'arrière-plan conformément à la nouvelle charte graphique du jeu et des logos. Les blocs de contenu ont été rendus moins opaques afin de mettre davantage en valeur l'image de fond. De plus, les boutons qui étaient auparavant indisponibles le sont désormais, car nous pouvons enfin publier les dernières ressources sur le site, y compris le rapport de projet et le projet lui-même.

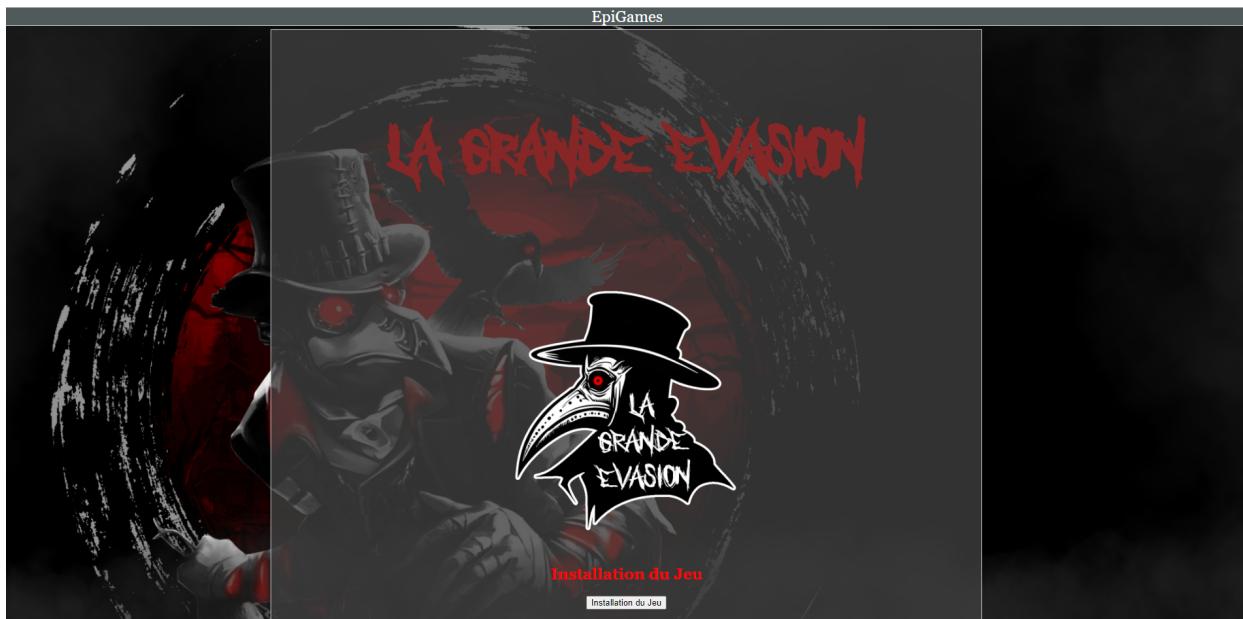


FIGURE 78 – Site La Grande Evasion

6 Conclusion

Le développement de notre jeu "La Grande Évasion" a été une entreprise complexe et enrichissante, intégrant divers éléments de design, de mécanique, d'intelligence artificielle, de son et d'interactivité. Chaque composant, de la prison immersive au personnage unique, du garde menaçant aux objets interactifs, a été soigneusement conçu et implémenté pour créer une expérience de jeu cohérente et captivante. Le mode multijoueur, essentiel pour la coopération, a été méticuleusement intégré, assurant une synchronisation fluide entre les joueurs.

Notre objectif principal était de créer un environnement immersif et engageant où les joueurs peuvent vivre une aventure palpitante en résolvant des énigmes et en évitant les dangers. Grâce à une combinaison d'efforts collaboratifs, d'ajustements minutieux et de tests rigoureux, nous avons réussi à atteindre cet objectif. Le jeu offre désormais une expérience fluide et intuitive, avec des mouvements naturels, des animations cohérentes, des énigmes stimulantes et une ambiance sonore immersive.

Les défis rencontrés tout au long du développement nous ont permis de repousser nos limites et d'apprendre continuellement, aboutissant à un produit final dont nous sommes fiers. "La Grande Évasion" incarne notre vision d'un jeu d'évasion immersif et coopératif, et nous sommes impatients de le partager avec vous pour qu'ils puissent eux aussi vivre cette aventure intense et mémorable.