Rapport de projet

Mai 2023

"STUCK"

. . . . . .

Akomi ZEBILA
Salim CHARIKH
Alexis GALOPIN
Paul MARIONNET

Groupe : **AVAC#**

EPITA Promo 2027

Table des matières

1 Introduction

Notre projet du S2 s'intitule *Stuck*. Il s'agit d'un survival horror créé à l'aide d'*Unity*. Dans ce jeu, le joueur incarne un personnage qui se réveille dans un manoir sinistre et lugubre. Sa mission est de capturer des rushs vidéo de monstres qui hantent les lieux tout en essayant de survivre. Le joueur dispose d'une caméra et doit filmer un maximum de séquences des monstres. Mais attention, ils sont imprévisibles et peuvent surgir à tout moment pour attaquer le joueur, d'autant plus que la caméra se décharge au fur-et-à-mesure de la partie.

Le mode solo offre une expérience immersive et terrifiante, mais nous avons également ajouté un mode multijoueurs pour ceux qui préfèrent jouer en coopération avec leurs amis. Dans ce mode, les joueurs doivent travailler ensemble pour survivre aux attaques des monstres et capturer le plus de rushs vidéo possible.

Au début du semestre, Vincent Carret a malheureusement quitté l'établissement et donc le groupe ; il était chargé principalement de la modélisation des personnages, de la réalisation du trailer et du site web, du gameplay ainsi que des animations. Nous avons donc dû revoir la répartition des tâches *(cf. second cahier des charges)*.

L'arrivée de Paul a également modifié les plans de bases, mais il a su s'adapter et prendre le train en marche, ce qui a beaucoup aidé l'avancée du projet.

2 Notre projet

2.1 Présentation de l'équipe AVAC#

Nous sommes ravis de vous présenter notre groupe de développement de jeux appelé "AVAC#". Composé initialement de Akomi, Vincent, Alexis et Salim Charikh (surnommé C# en référence au langage de programmation), nous avons travaillé en étroite collaboration pour créer un jeu d'horreur captivant. Cependant, au début de notre projet, Vincent a malheureusement quitté l'école, laissant un vide dans notre équipe. Cela a été de courte durée puisque Paul a été ajouté au groupe et a rapidement su s'intégrer et contribuer de manière significative au projet. Son expertise et son engagement ont été précieux pour mener à bien notre jeu. Grâce à nos compétences complémentaires, notre passion commune pour les jeux vidéos, nous avons pu relever les défis techniques et créatifs rencontrés tout au long du développement. Nous sommes fiers du résultat final et nous sommes impatients de partager notre jeu avec le public.

Salim CHARIKH
En tant que passionné de l'informatique, ce projet a été une expérience formidable dans le domaine du développement de jeux vidéo, où j'ai principalement travaillé sur l'IA et les animations.

Alexis GALOPIN
Issu d'une Terminale avec pour spécialités NSI et Mathématiques, j'ai été ravi de pouvoir travailler sur ce projet qui allie deux de mes plus grandes passions. Bien qu'ayant perdu des heures de sommeil et quelques cheveux, j'en ai appris énormément.

Akomi ZEBILA
Comme les autres membres du groupe, je suis passionné par l'informatique. Grâce à ce projet riche en expérience, j'ai pu en apprendre beaucoup sur le développement de jeux vidéos et plus particulièrement sur la conception de maps.

Paul MARIONNET
Passionné de l'informatique et grand joueur de jeux vidéo ce premier gros projet de jeu sur Unity m'a demandé beaucoup de patience mais m'a permis d'évoluer énormément et m'a appris de nombreuses choses.

2.2 Origine et nature du projet Stuck

2.2.1 Origine du projet

Nous avons d'office été d'accord de réaliser un jeu pour ce projet. Le premier choix était de décider s'il allait être en 2D ou 3D. Nous avons souligné un manque d'idées pour la première option, sachant que nous avions pour idée de créer une ambiance angoissante, ce qui est très difficile à instaurer dans ce type de jeu. Pour une immersion totale, un jeu de type FPS était évident.

Nous avons ainsi décider de faire un jeu qui saura lier l'aspect angoissant et l'aspect progression et où l'on pourra faire évoluer notre personnage au fur et mesure de la partie, ce sera donc un "Survival Horror" (abordé dans la section 3).

2.2.2 Concept du jeu

Le principe est assez simple : les joueurs (ou le joueur s'il est seul) incarnent des vidéastes passionnés de paranormal et sont bloqués dans un manoir réputé hanté dont ils doivent réussir à s'échapper. Pour corser les choses, deux types de monstres les en empêchent, chacun ayant sa particularité et forçant ainsi le(s) joueur(s) à adapter leur style de jeu. Le but du jeu est donc de s'enfuir sans se faire tuer par ceux-ci.

Pour pouvoir "s'échapper", les joueurs disposent d'une caméra avec laquelle ils doivent filmer les monstres ce qui a pour effet de remplir la barre de stockage de la caméra. Une fois son stockage plein, le joueur peut alors s'échapper.

Les ennemis seraient donc des IA chacune différentes. Pour ce qui est de la partie multijoueur, ce mode est identique (dans sa

globalité) au mode solo, à ceci près que les joueurs apparaissent à des lieux différents et peuvent se regrouper et s'entraider pour espérer augmenter leurs chances de gagner. Ils doivent cependant gagner chacun de leur côtés.

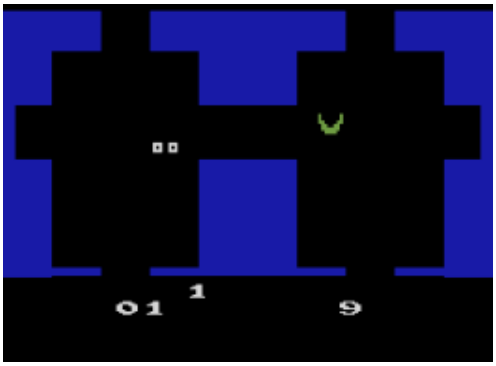
En plus de cela, la caméra a une batterie qui se vide au fur et à mesure que l'on filme. On peut cependant la recharger en ramassant des piles disposées partout dans la map.

2.3 Etat de l'art

Les prémices des jeux de survival horror remontent aux années 80-90 où ces derniers s'inspirent des fictions d'horreur les plus anciennes.

2.3.1 Haunted House (1981)

Un de ces jeux considéré comme l'un des précurseurs du genre est Haunted House, sorti en 1981 sur ATARI 2600. L'unique joueur incarnait une paire d'yeux qu'il devait faire naviguer dans un manoir hanté pour récupérer les trois morceaux d'une urne.



2.3.2 Alone in the dark (1992)

En 1992, le groupe français "Infogrames" développe Alone in the Dark, qui sera considéré comme le père des Survival Horror. Il met en scène un protagoniste solitaire affrontant des hordes de monstres, devant résoudre des énigmes et trouver des clés cachées dans un environnement 3D pré-calculé.

2.3.3 Granny (2017)

Plus récemment, on retrouve des jeux comme Granny, un jeu où l'on se réveille captif dans une maison piégée de laquelle on doit s'échapper grâce à divers outils et énigmes.

2.3.4 Bigfoot (2017)

BIGFOOT est un jeu de survie et d'horreur où l'on doit chasser le "Bigfoot", une créature légendaire, sans se faire attraper. Il y a

plusieurs modes de jeux, il y a le mode solo où l'on incarne soit un humain et on devra survivre, ou bien le Bigfoot où au contraire on devra chasser l'humain. Et il y a le mode multi ou l'on doit essayer de survivre en groupe.

2.3.5 SCP : Secret Laboratory (2017)

Enfin, le dernier jeu (dont nous souhaitons nous inspirer pour le fonctionnement du multijoueur). Il s'agit d'un jeu en multi développé par Northwood Studios en 2017 et sorti tout droit de l'univers de la "Fondation SCP", une œuvre de fiction tirée du site d'écriture collaborative du même nom. À chaque partie, les joueurs ont des classes attribuées et devront, en fonction de cette dernière, s'échapper du laboratoire et éviter les monstres, ou au contraire éliminer ceux qui tentent de fuir.

Ces jeux partagent plusieurs points communs qui font qu'on peut les classer dans ce genre "Survival Horror" comme par exemple :

- l'usage de créatures issues de fictions d'horreur qui ont chacun un comportement unique dans le jeu.

- la présence d'énigmes à résoudre qui rajoute de la difficulté à la tâche.

- une atmosphère angoissante (d'où "Horror").

- la nécessité de rester en vie (d'où le "Survival" et parfois de s'échapper d'un endroit clos.

- la possibilité d'équiper des objets aidant le joueur dans sa tâche
  (armes,protection,communications,etc...).

Nous nous sommes appliqués à implémenter certaines de ces mécaniques pour offrir une expérience qui respecte le genre tout en y mettant notre propre pâte.

3 Découpage du projet

3.1 Répartition des tâches

Suite à l'arrivée de Paul dans le groupe, nous avons revu la répartition des tâches pour pouvoir l'inclure comme il se faut. Il est venu aider Alexis sur le développement du site et des interfaces graphiques, et s'est chargé de la connexion du joueur.

Voici la version finale du tableau de répartition des tâches :

Tâches	Akomi	Salim	Alexis	Paul
Gameplay		◯	⊗	
Interfaces Graphiques			◯	⊗
Modélisation personnages/objets	◯	⊗		
Animations	⊗	◯		
Modélisation décors (map)	⊗		◯	
Réseau			◯	⊗
IA	◯	⊗		
Site Web			⊗	◯

Légende :
⊗ -> Responsable ◯ -> Suppléant

3.2 Déscription des tâches

Nous avons fait des progrès significatifs dans l'amélioration de notre intelligence artificielle. Elle est désormais capable de se déplacer librement dans la carte et même de nous attaquer lorsque nous sommes à portée. De plus, nous avons réussi à synchroniser les animations de nos différents modèles avec les scripts d'IA correspondants, ce qui nous a contraints à développer un script spécifique pour chaque modèle en raison de leurs caractéristiques et animations distinctes. Ces améliorations ont permis d'ajouter une dimension réaliste et immersive à notre jeu, offrant une expérience plus gratifiante et immersive pour les joueurs. Cependant, il convient de noter que l'IA n'est pas encore tout à fait implémentée et qu'il reste plusieurs paramètres à régler, tels que la vitesse, la précision(Réaction lors d'un sifflement par exemple) et la réactivité de l'IA. Nous comptons bien évidemment améliorer cela pour le soutenance finale.

3.3 Avancement du projet

Voici le planning d'avancement qui reflète de manière globale notre progression sur les différentes tâches lors des différentes soutenances.

Soutenance	1	2	3
Gameplay	40%	60%	100%
Interfaces Graphiques	30%	70%	100%
Modélisation personnages/objets	5%	60%	100%
Animations	0%	35%	100%
Modélisation décors (map)	33%	60%	100%
Réseau	40%	60%	100%
IA	40%	60%	100%
Site Web	30%	75%	100%

3.4 Outils et logiciels utilisés

Nous avons continué à utiliser Git et GitHub pour faciliter la gestion du code source et assurer une coordination fluide entre les membres de l'équipe. Notre serveur Discord a également été un outil précieux pour maintenir une communication régulière et organiser les différentes tâches de manière efficace. Nous comptons faire tester le jeu à nos amis qui sont d'ailleurs présents sur le serveur Discord pour avoir des retours extérieurs.

3.4.1 Discord

Dans le cadre de notre projet de création du jeu "Stuck" sur Unity, nous avons mis en place une structure de communication efficace en utilisant Discord comme plateforme principale. Discord nous a permis de rassembler tous les membres de l'équipe dans un même espace virtuel, favorisant ainsi une collaboration fluide et une communication constante.

Nous avons créé un serveur Discord dédié spécifiquement à notre projet, offrant une variété de canaux et de salons pour organiser nos discussions de manière ordonnée. Le point central de notre serveur est un salon privé réservé exclusivement aux membres du projet. Ce canal privé nous permet d'échanger des informations confidentielles, de partager des documents importants, de planifier des réunions et de prendre des décisions stratégiques en tant qu'équipe.

En plus du canal privé, nous avons également mis en place des salons généraux accessibles à des personnes extérieures à l'équipe du projet. Ces salons nous permettent d'inviter des testeurs et des joueurs potentiels afin qu'ils puissent expérimenter notre jeu et partager leurs commentaires et suggestions. Cette interaction

avec des personnes extérieures nous aide à obtenir des points de vue différents et à améliorer la convivialité et la qualité globale de notre jeu.

En utilisant Discord, nous avons pu bénéficier de fonctionnalités telles que la possibilité de partager des captures d'écran, des vidéos et des fichiers, facilitant ainsi la communication visuelle et la rétroaction précise. De plus, les fonctionnalités de discussion en temps réel de Discord nous ont permis de réagir rapidement aux questions, aux préoccupations et aux défis rencontrés lors du développement du jeu.

En résumé, Discord a joué un rôle essentiel dans l'organisation et la coordination de notre projet de jeu "Stuck". Il nous a fourni un espace centralisé pour communiquer, partager des informations, solliciter des retours d'utilisateurs et prendre des décisions collectives. Grâce à cette plateforme, nous avons pu maintenir une collaboration efficace et obtenir des commentaires précieux, ce qui a contribué au succès de notre projet.

3.4.2 Git

Nous avons utilisé Git et GitHub pour gérer notre code source, bien que nous ayons travaillé sur une seule et même branche. Git nous a permis de suivre les modifications apportées au code, de conserver un historique des versions et de collaborer de manière organisée.

Bien que l'utilisation de branches distinctes soit généralement recommandée pour faciliter la gestion des modifications parallèles et minimiser les conflits, il est possible de travailler sur une seule branche si l'équipe est en mesure de collaborer de manière coordonnée. Dans notre cas, nous avons choisi de travailler sur une seule branche afin de simplifier le processus de développement et

de maintenir une synchronisation étroite entre les membres de l'équipe.

Git nous a permis de suivre les modifications apportées au code de manière granulaire, en enregistrant chaque commit avec des commentaires significatifs. Cela nous a permis de revenir à des versions antérieures du code si nécessaire et d'identifier plus facilement les modifications responsables des problèmes rencontrés.

L'utilisation de GitHub comme plateforme d'hébergement pour notre dépôt Git nous a permis de partager notre code source avec les membres de l'équipe, de faciliter la collaboration et de garder une trace des modifications apportées. Bien que nous ayons travaillé sur une seule branche, GitHub nous a permis de créer des problèmes et des tâches pour organiser notre travail, de discuter des problèmes spécifiques du code et d'effectuer des révisions par le biais de pull requests.

En résumé, bien que nous ayons travaillé sur une seule branche, l'utilisation de Git et GitHub nous a permis de gérer notre code source de manière structurée et de faciliter la collaboration entre les membres de l'équipe. Cela nous a aidés à suivre les modifications apportées au code, à maintenir un historique clair et à garantir la cohésion de notre travail collectif.

4 Avancement détaillé du projet

4.1 Réseau

Mise en place du réseau et création des lobbys : La première étape consistait à mettre en place le réseau et à concevoir une structure permettant la création de lobbys offrant une expérience de jeu plaisante. Une recherche approfondie a été effectuée pour documenter les meilleures pratiques. Ensuite, les fonctions de base ont été développées et Unity ainsi que le cloud ont été configurés pour prendre en charge les lobbys.

Fonctions de base et configuration : Les fonctions de base comprenaient la création des lobbys, la récupération d'informations et de mises à jour concernant les lobbys, ainsi que la possibilité de rejoindre des lobbys à l'aide de codes spécifiques. Ces fonctions ont été implémentées pour répondre aux besoins spécifiques du projet.

Affichage des joueurs en temps réel : Une exigence clé était d'afficher en temps réel les joueurs présents dans le lobby. Pour cela, une méthode a été développée pour afficher des préfabs de joueurs en fonction du nombre de joueurs présents dans le lobby. Ces informations ont été récupérées en utilisant les fonctions précédemment créées.

Résolution des problèmes techniques : Durant le développement, plusieurs bugs ont été identifiés, notamment en raison des limitations de requêtes imposées par l'API. Pour résoudre ce problème, les requêtes d'actualisation des informations du lobby ont été espacées, ce qui a affecté la réactivité du système. Des efforts ont été déployés pour optimiser ces requêtes et améliorer les performances globales.

4.2 IA

La mise en place d'une IA pour un jeu d'horreur est essentielle pour créer une expérience immersive. L'IA gère le comportement des monstres, ce qui est crucial pour le gameplay du jeu. Les monstres sont programmés pour suivre le joueur, ce qui crée une immersion en le mettant sous pression.

Une fonctionnalité intéressante a été ajoutée à l'IA, permettant aux monstres de suivre le sifflement du joueur. Cela peut être utilisé pour les attirer dans des pièges ou pour créer des moments de tension lorsque le joueur est en danger.

Le Navmesh est utilisé pour définir les zones où l'IA peut se déplacer dans le jeu. En ajoutant un composant Navmesh Agent aux monstres, l'IA peut se déplacer en douceur sur la carte et interagir avec les éléments du jeu.

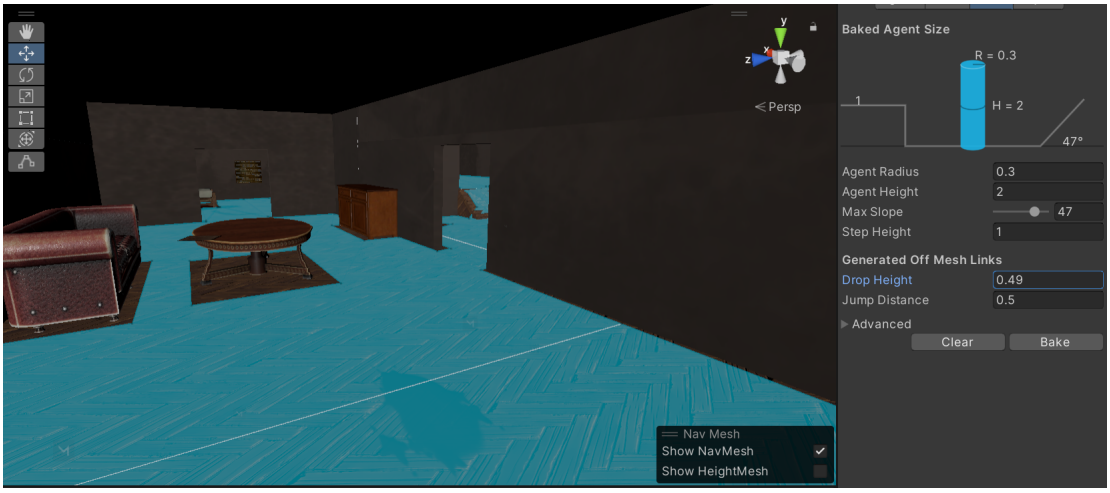


FIGURE 1 – Exemple d'une surface NavMesh sur un des étages de la map

De plus, un composant NavMeshObstacle a été ajouté, permettant de créer des obstacles mobiles ou temporaires dans le NavMesh. Cela bloque la navigation des agents et permet de créer des situations de gameplay intéressantes où les chemins sont modifiés dynamiquement.

Le script ClownController est utilisé pour détecter la présence du joueur et le suivre. Des conditions sont mises en place pour empêcher les monstres de voir à travers les murs ou de suivre le joueur après avoir entendu un sifflement, par exemple.

L'IA des monstres peut également attaquer le joueur, créant ainsi une sensation de peur et de tension caractéristique des jeux d'horreur. Cependant, le joueur a la possibilité de ralentir les monstres.

Chaque script d'IA gère des animations différentes, car chaque modèle se comporte différemment. Des variables booléennes peuvent être ajoutées à l'onglet de l'animateur pour déclencher les transitions entre les animations. Ces variables sont définies comme "True" ou "False" à partir du script, permettant de lancer des

FIGURE 2 – Images montrant le clown en train d'attaquer le joueur

animations spécifiques.

En résumé, l'IA joue un rôle crucial dans la création d'une expérience immersive dans un jeu d'horreur. Elle gère le comportement des monstres, leur permettant de suivre le joueur et d'ajouter des fonctionnalités telles que la détection du sifflement. Le Navmesh et le script ClownController sont utilisés pour faciliter les déplacements des monstres et définir des conditions spécifiques à leur comportement.

4.3 Gameplay

4.3.1 Bases

Dans les choses les plus rudimentaires de l'implémentation du gameplay, on retrouve les mouvements du joueur. Assez simple d'usage, le joueur peu se déplacer à 3 vitesses différentes, une lente lorsqu'on appuie sur Controle, une moyenne (celle de base) et une rapide lorsqu'on appuie sur Shift gauche. Le joueur peut également sauter avec Espace. A l'aide de la fonction 'Vector2.SmoothDamp', nous avons rendu ces mouvements plus fluides et agréables. Nous avons de la même manière fluidifié les mouvements de caméras.

4.3.2 Score & Vie

La vie est tout ce qu'il y a de plus simple, lorsqu'un joueur n'a plus de point de vie, il meurt et perd la partie. Pour le score, un fichier CameraRaycast.cs et Score.cs ont été créés. Le premier projette un "champ" de plusieurs Raycasts qui détectent si le monstre est dans le champ de vision du joueur, le second, lui, vérifie que la caméra est allumée et que CameraRaycast.cs détecte le monstre auquel cas l'espace de stockage (qui fait office de score) de la caméra augmente. Une fois le stockage plein, le joueur a gagné. Le score augmente d'un point (symbolique) toutes les 3 secondes et il en faut 25 pour gagner, soit 1 min 15 à filmer le monstre.

4.3.3 Interactions

Pour ce qui est des interactions, le joueur peut : siffler à l'aide de la touche X qui déclenche un son de sifflement aléatoire parmi trois audios ; ouvrir les portes à proximité avec F ; allumer sa caméra avec le Clique Droit ; ramasser et lâcher des piles avec le Clique

Gauche ; recharger la caméra avec R et allumer ou éteindre sa torche avec E.

Chacune de ces interactions a un script permettant de la gérer. Pour les portes, un Raycast est tiré pour détecter si on est assez proche d'une, puis on vérifie si le joueur appuie sur F. Pour les autres interactions, une simple détection de la touche associée est faite avec la méthode GetKeyDown.

4.3.4 Caméra

Lors de son utilisation la caméra perd de la batterie. Cela est réalisé via le script CameraBattery.cs qui décrémente d'un tiers de sa batterie la caméra toutes les 30 secondes. Une fois vide, la caméra ne peut plus être utilisée tant qu'elle n'est pas rechargée. Si le joueur dispose de piles d'en son inventaire, il peut appuyer sur R pour recharger la caméra d'1/3.

4.4 Modélisation personnages et animation

Notre processus de création de modèles a été grandement facilité grâce à l'utilisation de *Poly.pizza* et *Mixamo.com*. Nous avons utilisé ces ressources gratuites pour sélectionner les modèles correspondant à nos besoins, puis les avons personnalisés en utilisant le logiciel *Blender*. Grâce à *Mixamo*, nous avons également pu importer des modèles et les animer facilement en attribuant des zones spécifiques du modèle à des parties du corps correspondantes. Cette fonctionnalité nous a permis d'appliquer de manière fluide toutes les animations proposées par *Mixamo* sur nos modèles, donnant vie à nos personnages de jeu de manière plus réaliste et immersive. De plus, pour enrichir encore davantage l'expérience de jeu, nous avons utilisé l'outil Animator de Unity. Cet

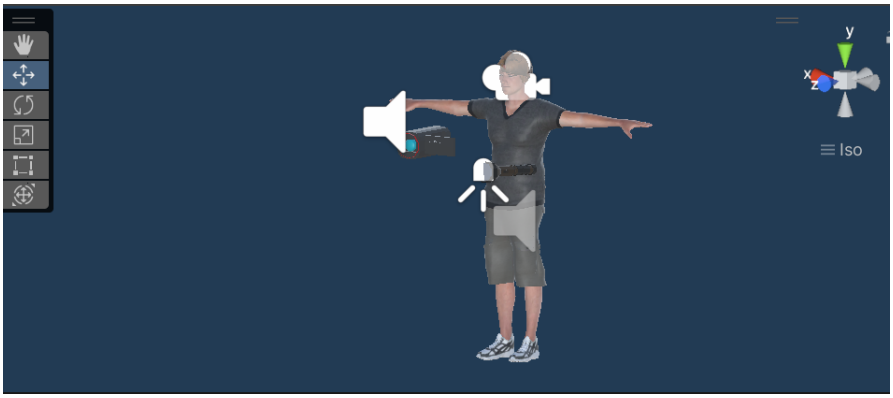


FIGURE 3 – Le prefab du joeur Rémy

outil nous a permis de paramétrer et de relier différentes animations pour créer une architecture complète d'animations. Nous avons ainsi pu créer des animations plus complexes et fluides pour nos personnages de jeu. En utilisant l'outil Animator, nous avons pu contrôler les transitions entre les différentes animations et ajouter des effets spéciaux pour améliorer l'expérience utilisateur.

4.4.1 Rémy

Rémy est le personnage principal du jeu, et c'est le personnage que le joueur contrôle. Son modèle a été directement importé de Mixamo, y compris ses animations. Cependant, l'animation d'un personnage jouable en 3D nécessite une approche légèrement différente de celle habituellement utilisée. Au lieu d'utiliser des transitions classiques d'un Animator avec des booléens, nous allons utiliser un "Blend Tree" dans lequel nous pouvons ajouter des directions. Pour notre animation, nous utiliserons uniquement les axes "X" et "Y" pour représenter l'accélération du joueur. Le "Blend Tree" sera relié à des animations, et pour chacune de ces animations, nous allons définir les valeurs "X" et "Y" qui déclen-

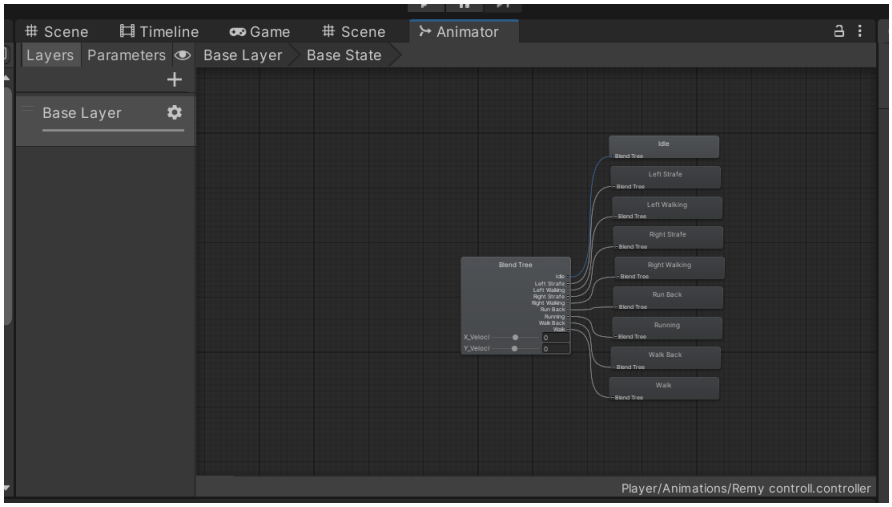


FIGURE 4 – Exemple de Blend Tree sur lo modèle Rémy

cheront l'animation correspondante. Ainsi, les transitions se feront automatiquement et de manière fluide.

Pour donner une logique à tout cela, nous devrons attacher un script qui permettra de récupérer les vitesses et la direction du joueur, afin de déterminer comment l'animation sera déclenchée. L'état de base du joueur sera représenté par l'animation "Idle", qui sera jouée lorsque le joueur ne se déplace pas.

En utilisant cette approche, nous pourrons animer Rémy de manière réaliste et fluide, en fonction de ses mouvements et de ses actions dans le jeu. Cette méthode permettra au joueur de se sentir davantage immergé dans l'expérience de jeu, car les animations s'adapteront de manière naturelle aux actions du personnage.
Rémy dispose également d'une caméra ainsi que d'une lampe torche, deux éléments essentiels au gameplay.

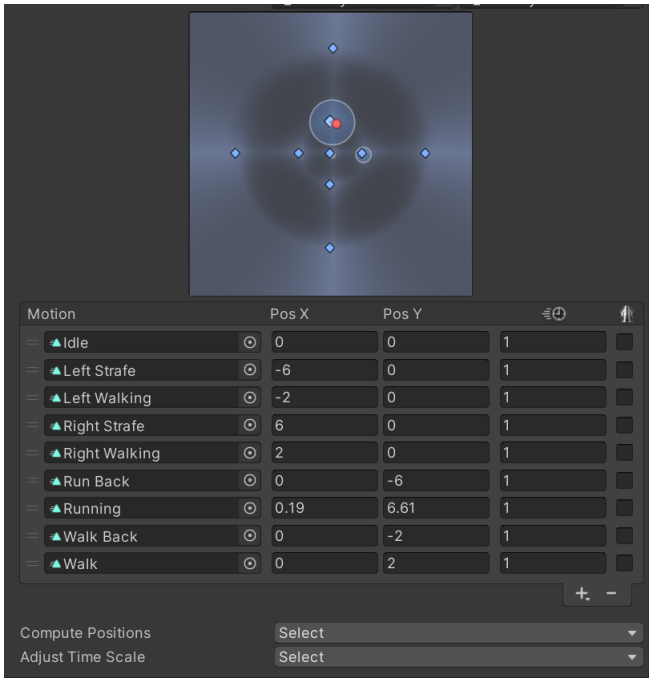


FIGURE 5 – Tableau permettant de gérer les animation du Blend Tree

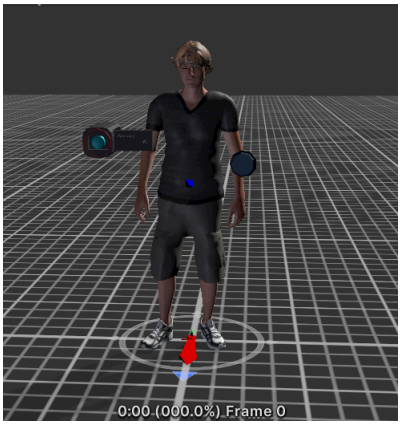


FIGURE 6 – Rémy en Idle(état de base)

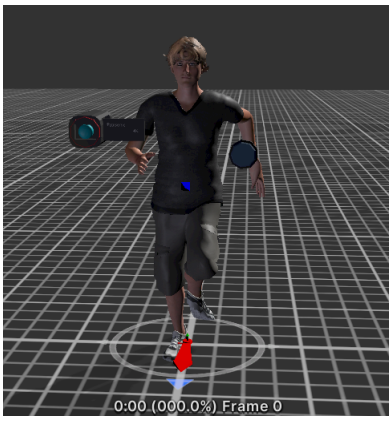


FIGURE 7 – Rémy en train de courir

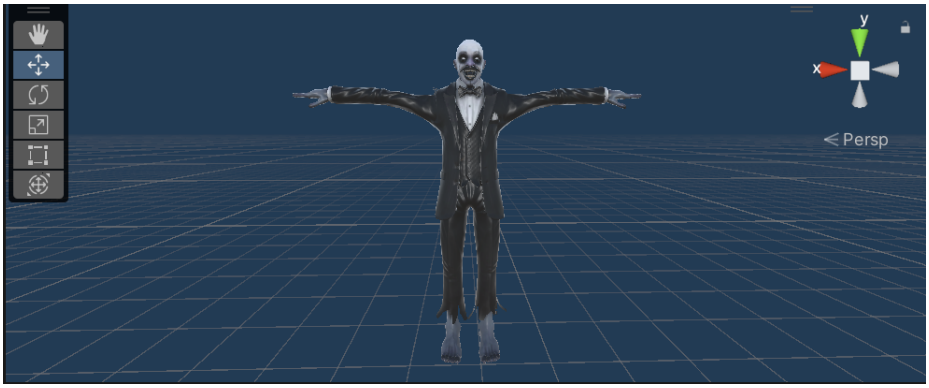


FIGURE 8 – Le modèle du clown

4.4.2 Le clown

Dans notre jeu, le clown joue le rôle du monstre résidant dans le manoir, et sa présence est cruciale pour pouvoir s'échapper. Dès que les joueurs pénètrent dans le manoir, ils perturbent le sommeil du clown, déclenchant ainsi sa fureur et son instinct de traque envers les intrus.

Pour donner vie au clown, nous avons utilisé des ressources fournies par Mixamo, notamment son modèle et ses animations. Nous avons opté pour des animations "classiques" qui correspondent au comportement typique d'un clown effrayant. Grâce à l'onglet "animator" d'Unity, nous avons créé différentes transitions entre les animations, permettant ainsi des mouvements fluides et réalistes pour le clown.

Ces transitions entre les animations sont gérées par des scripts dans le jeu. Nous avons programmé des scripts qui détectent les conditions appropriées pour déclencher chaque transition d'animation, en fonction des actions du joueur ou de l'état du clown lui-même. Par exemple, lorsque le clown repère le joueur, une transition d'animation se déclenche pour le faire passer de son comportement normal à celui de traque.

L'utilisation de Mixamo et les capacités de l'onglet "animator" d'Unity nous ont permis de donner vie au clown en lui fournissant des animations fluides et réactives. Les scripts associés ont permis de contrôler les transitions entre ces animations, créant ainsi une expérience immersive et effrayante pour les joueurs.

En résumé, dans notre jeu "Stuck", le clown joue le rôle du monstre du manoir. Grâce à Mixamo, nous avons utilisé son modèle et ses animations pour donner vie à ce personnage. En utilisant l'onglet "animator" d'Unity, nous avons créé des transitions entre différentes animations du clown, gérées par des scripts, afin

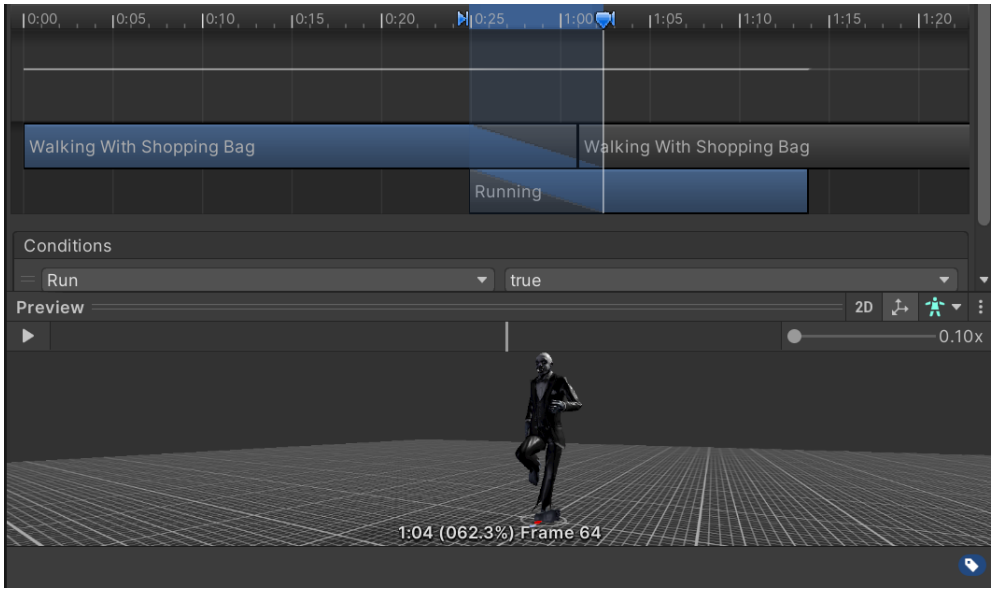


FIGURE 9 – Exemple de transition

de lui conférer des comportements réalistes et effrayants. Cela contribue à l'expérience immersive et angoissante du jeu pour les joueurs.

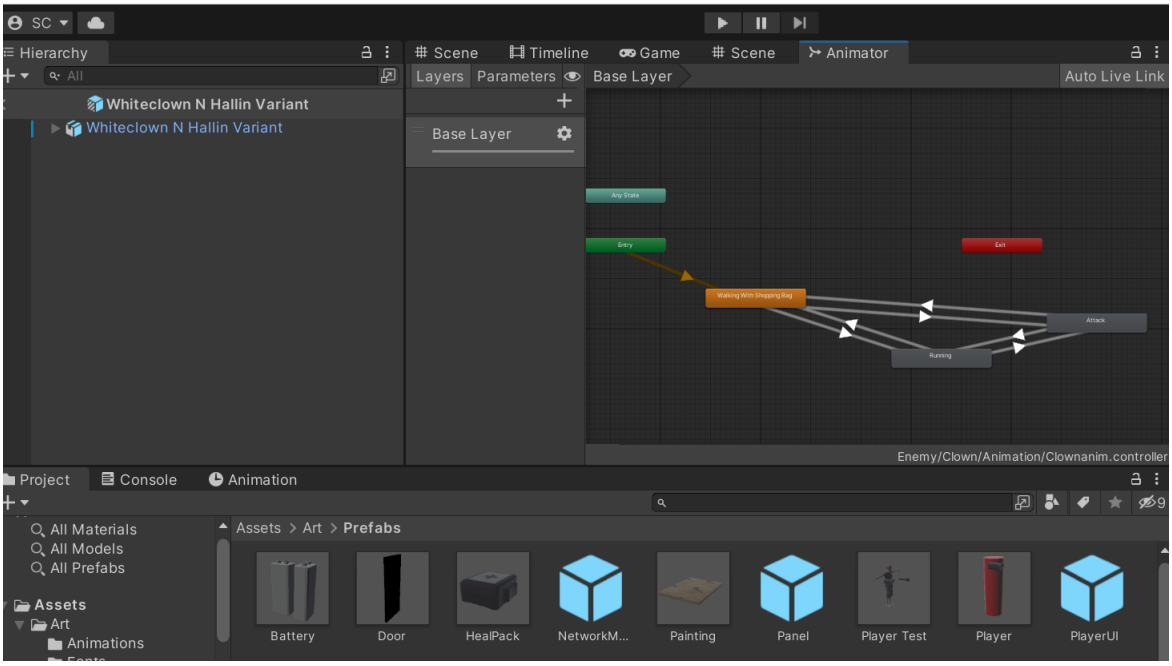


FIGURE 10 – Onglet Animator

4.4.3 Les portes

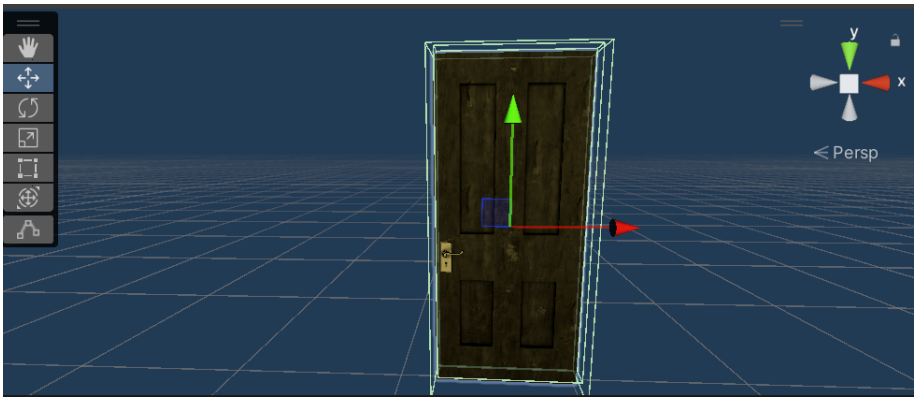


FIGURE 11 – Le prefab de la porte

L'asset "Door" fait partie d'un pack d'actifs payant disponible sur l'Unity Asset Store. Ce pack comprend une variété de ressources prêtes à être utilisées dans nos projets de jeu,
y compris des modèles de portes avec des animations préétablies.

Cependant, après avoir examiné attentivement les animations fournies, nous avons constaté qu'elles ne répondaient pas parfaitement à nos besoins spécifiques. Nous avions une vision précise de l'animation de la porte que nous souhaitions réaliser, afin de correspondre au style et à l'ambiance de notre jeu.

Pour créer notre propre animation, nous avons suivi une méthode couramment utilisée dans Unity. En ouvrant l'onglet "Animation" de la porte, nous avons pu accéder à un ensemble d'outils et de fonctionnalités pour manipuler et définir l'animation.

L'un des avantages de cette méthode était la possibilité de définir la durée de l'animation selon nos besoins. En ajustant les paramètres, nous pouvions choisir la longueur optimale pour que l'animation de la porte soit fluide et s'intègre harmonieusement au reste du jeu.

Un aspect pratique était la possibilité de définir les positions et rotations de la porte aux deux extrémités de l'animation. En

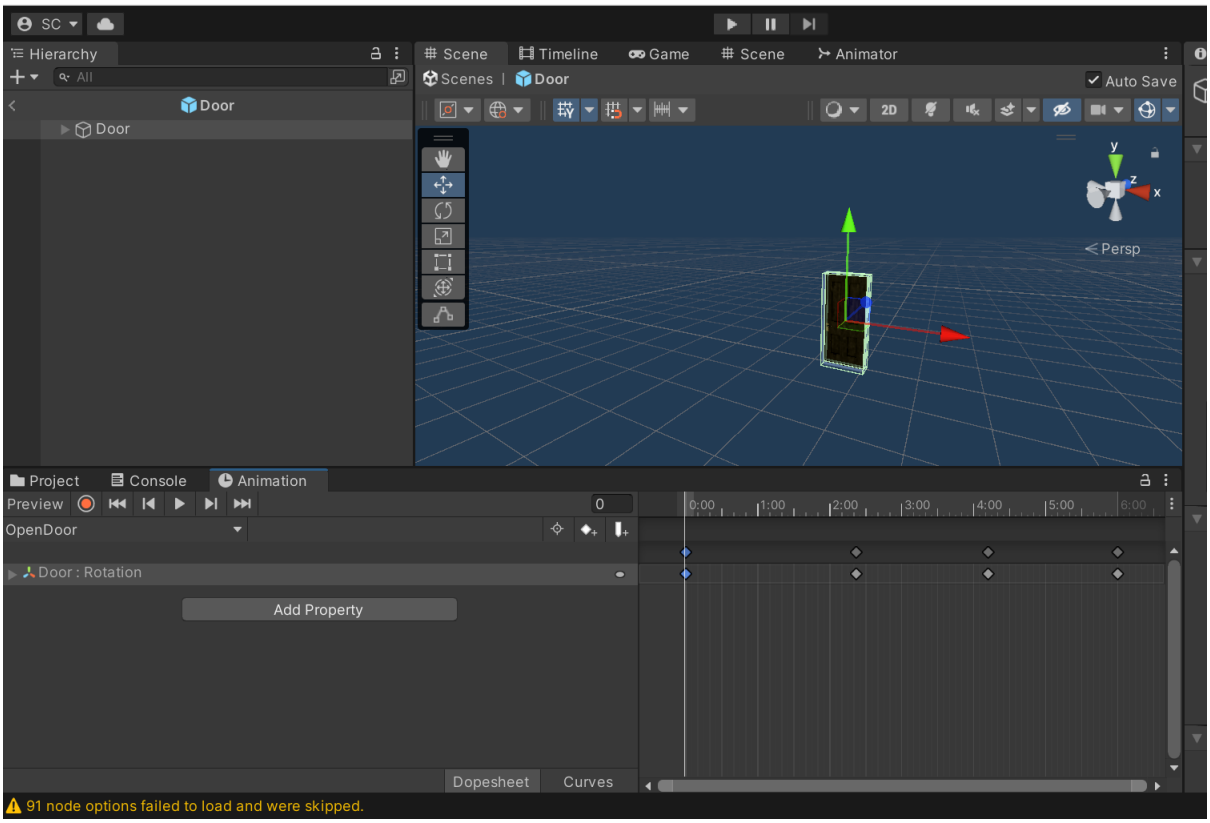


FIGURE 12 – L'onglet animation sur unity

spécifiant ces valeurs, nous avons pu créer notre propre séquence d'ouverture et de fermeture de porte, avec les mouvements précis que nous désirions.

Cette flexibilité nous a permis de personnaliser l'animation de la porte selon nos besoins artistiques et de design. En adaptant les mouvements, la vitesse et les angles de rotation, nous avons pu créer une animation qui correspondait parfaitement à notre vision du comportement de la porte dans le jeu.

En résumé, bien que l'asset "Door" fournissait des animations préexistantes, nous avons décidé de créer notre propre animation en utilisant les fonctionnalités d'animation disponibles dans Unity.

Cette approche nous a donné un contrôle total sur la durée, les positions et les rotations de la porte, nous permettant ainsi de personnaliser l'animation pour correspondre exactement à nos besoins spécifiques.

4.5 HUD et menus

Cette partie est l'autre partie qui prend le plus de code C# après le gameplay.

4.5.1 Le canvas PlayerUI

PlayerUI est le Prefab d'un canvas à l'origine de tout ce qui est interface graphique, HUD, etc... Il contient tous les autres menus et interfaces, c'est à dire :

- L'HUD de base ("HUD"), avec la barre de vie et le réticule.

- Le menu pause ("PauseMenu") qui permet d'accéder aux paramètres et quitter le jeu. Il affiche également l'état de la batterie de la caméra.

- Le menu des paramètres, dans lequel on peut changer la sensibilité ainsi que le volume des sons du jeu.

- L'interface de la caméra ("CamHUD") avec l'affichage de la batterie restante ainsi que de la barre de stockage et quelques éléments qui imitent l'écran d'une caméra. Cette interface est assez différent des autres (notamment il y a un filtre légèrement vert transparent) car elle est le seule élément qui permet au joueur de faire la distinction entre la vue du joueur et la vue à travers sa

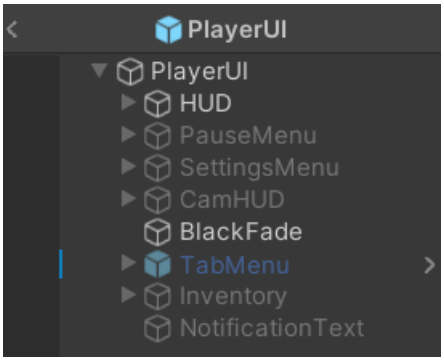


FIGURE 13 – Hiérarchie du prefab PlayerUI

caméra.

- L'inventaire ("Inventory") qui affiche le nombre de piles possédées.

- Un champ de texte ("NotificationText") qui permet d'afficher au joueur une notification/information depuis n'importe quel autre script puisque la méthodes Notify est static.

- Une image noire ("BlackFade") qui, en faisant varier sa transparence, sert de transition entre l'HUD et l'interface de la caméra.

4.5.2 Les scripts

Si beaucoup d'éléments disposent de leurs propres scripts, tel que les barre de vie/stockage ou les boutons, tout ce beau monde est principalement géré par le script PlayerUI.cs. C'est dedans qu'est géré entre autre la mise en pause du jeu ainsi que l'activation et désactivation des différents menus permettant de passer de l'un à l'autre. Pour pouvoir afficher du texte au joueur à n'importe quel moment, Alexis a eu l'idée de faire une fonction static,

plus précisément une coroutine. Ainsi on peut afficher du texte au joueur depuis n'importe quel script et à l'aide de l'IEnumerator WaitForSeconds on peut le faire pendant X secondes sans bloquer le reste du jeu.

```
public static IEnumerator Notify(string message, float displayTime)
{
    notification.SetActive(true);
    if (notification.GetComponent<TextMeshProUGUI>().text != message)
    {
        notification.GetComponent<TextMeshProUGUI>().text = message;
        yield return new WaitForSeconds(displayTime);
        notification.SetActive(false);
        notification.GetComponent<TextMeshProUGUI>().text = "";
    }
}
```

FIGURE 14 – Coroutine Notify

4.5.3 Scènes d'accueil

Deux scènes font offices de menus en dehors du jeux. La première ("HomeScene") sur laquelle le jeu se lance, permet de choisir les réglages et de lancer une partie, en plus d'accueillir le joueur. Il y a également un bouton si le joueur désire quitter le jeu. En cliquant sur Play, le joueur est redirigé vers la deuxième scène ("LoginScene") où le joueur est invité à rentrer un pseudonyme. Dans cette scène, il y a deux options pour le joueur, la première est l'option de créer un lobby. En cliquant sur cette option, le joueur créé donc un lobby y rentre et peut ensuite y voir le code en haut au milieu de l'écran, ce code peut être partagé avec d'autres joueurs afin de pouvoir rentrer dans ce même lobby. Sinon, le joueur a l'option de rejoindre un lobby avec un code. Une fois dans un lobby, soit après l'avoir créé soit après l'avoir rejoins avec un code, le joueur peut décider de rentrer dans le jeu en cliquant sur le bouton "Start Game".

4.6 Modélisation décors (map)

4.6.1 Assets utilisés

On a choisi d'utiliser différents types d'assets, que ce soit en format lowpolygons ou bien en format normal. Bien que cela ne corresponde pas forcément très bien, nous avons essayé de garder des modèles qui vont bien ensemble. Nous avons principalement téléchargé des assets de meubles, avec par exemple Old Kitchen Assets pour les meubles de cuisine, Old Military Bed et Small furniture pack pour les meubles des chambres, etFree Furniture Set, Furniture FREE Pack, Rustic Small Cabinet pour les meubles des couloirs et de la salle à manger. Les autres assets ont servi à créer l'atmosphère d'horreur. Nous avons tout d'abord acheté le pack Sinister Props Pack 1 au prix de 11.99 euros, qui contient des meubles et des objets sinistres tels que des armes (hache, scie, marteau) ou encore des matelas ensanglantés qui ont été disposés dans certaines chambres. Ce pack nous a également permis de placer des traces de sang aux quatre coins du manoir, ce qui bénéficie grandement à l'atmosphère d'horreur. Enfin, nous avons utilisé l'asset Steel Window pour recouvrir les fenêtres. Enfin, nous avons utilisé différents materials et textures pour décorer les murs, sols et plafonds. Les sols ont principalement été revêtus de parquet ou carrelage (dépendant de la pièce), les murs de peinture marron foncé, de même pour les plafond. Pour ce qui est du sous-sol, nous avons tout recouvert de pierres.

FIGURE 15 – Salle à ordures

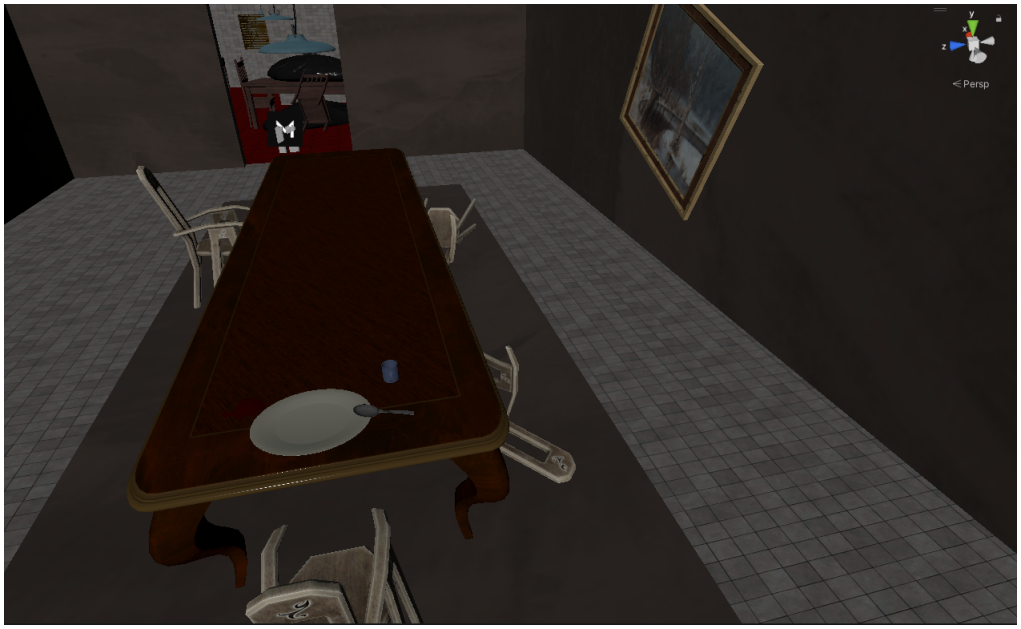


FIGURE 16 – Salle à manger

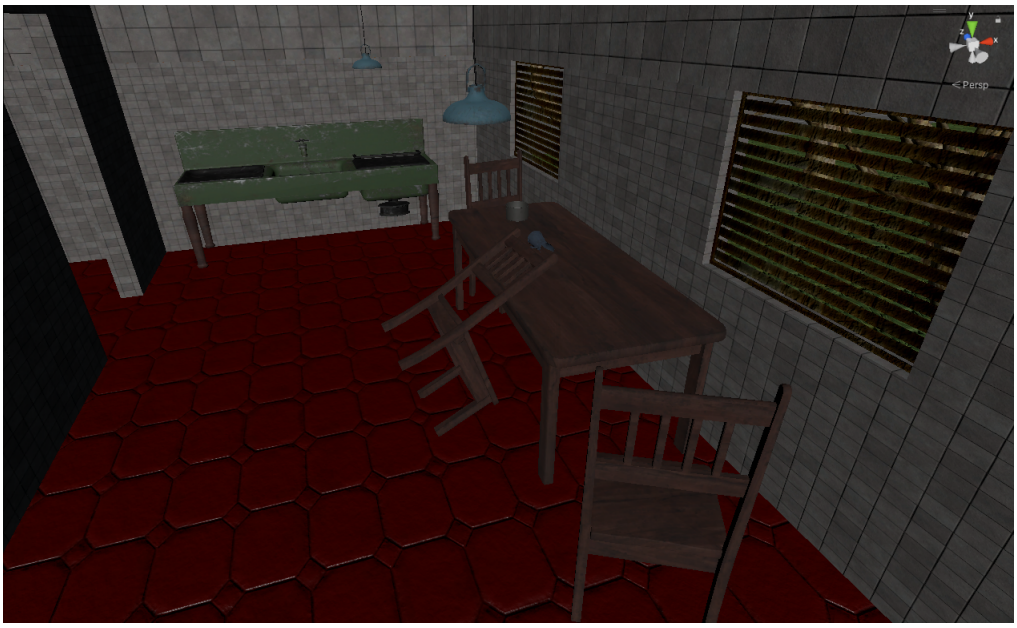


FIGURE 17 – Cuisine

FIGURE 18 – Salle de bain (rez-de-chaussée)

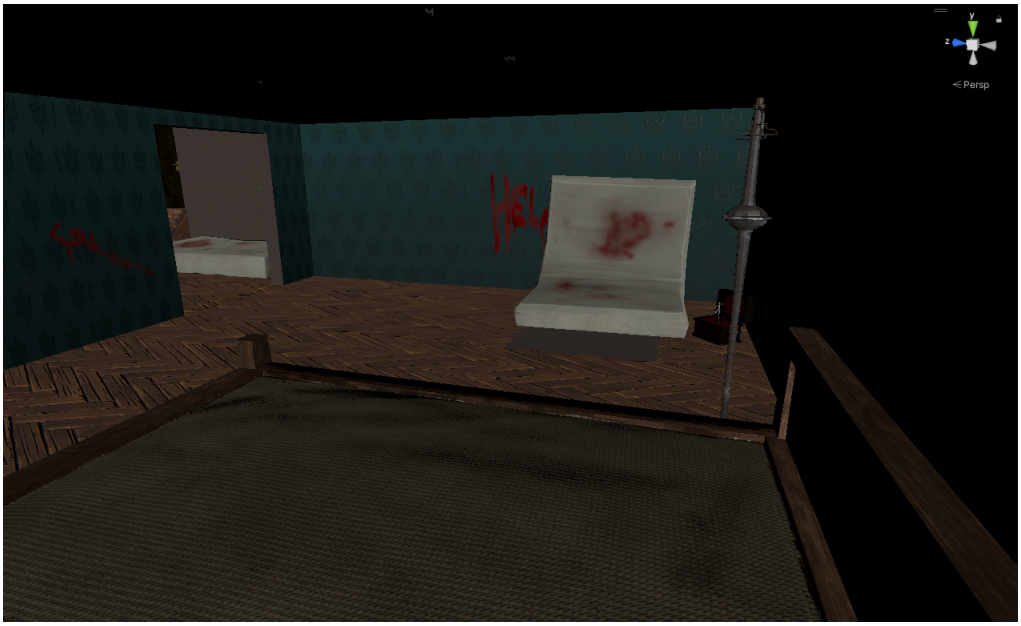


FIGURE 19 – Chambre 1

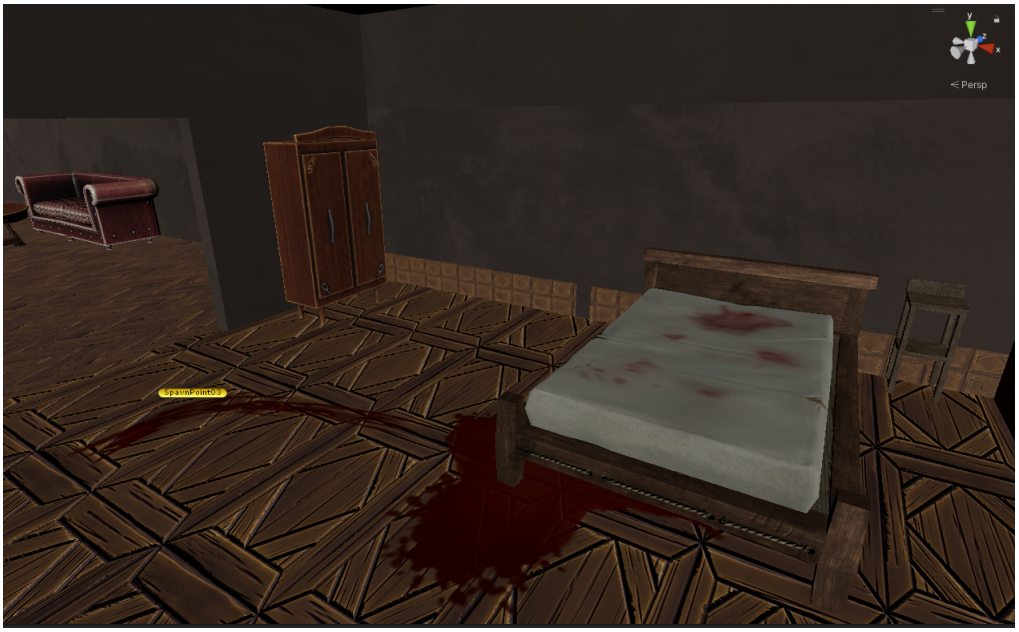


FIGURE 20 – Chambre 2

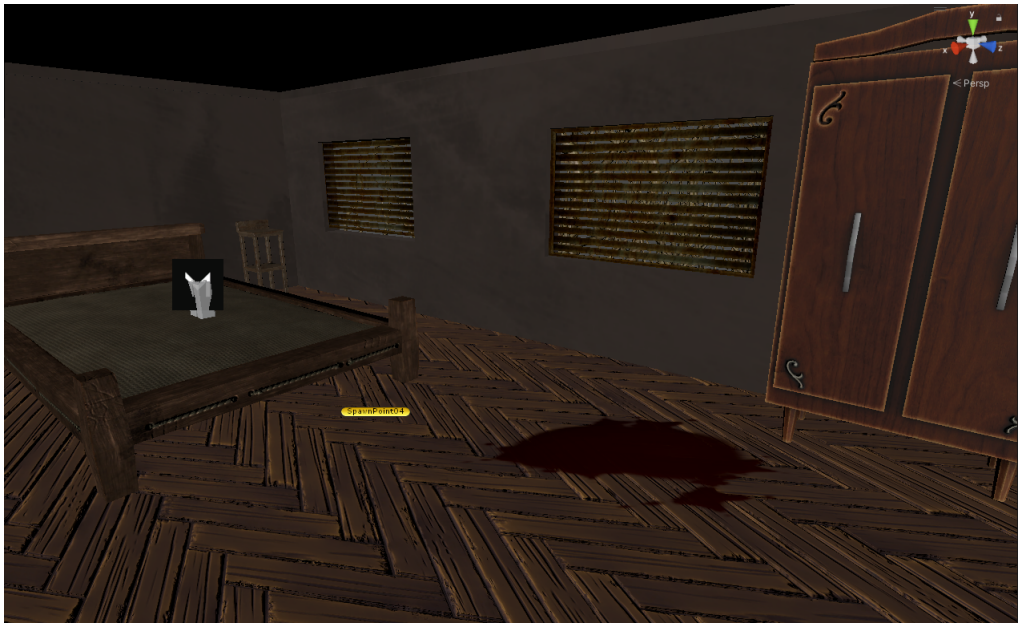


FIGURE 21 – Chambre 3

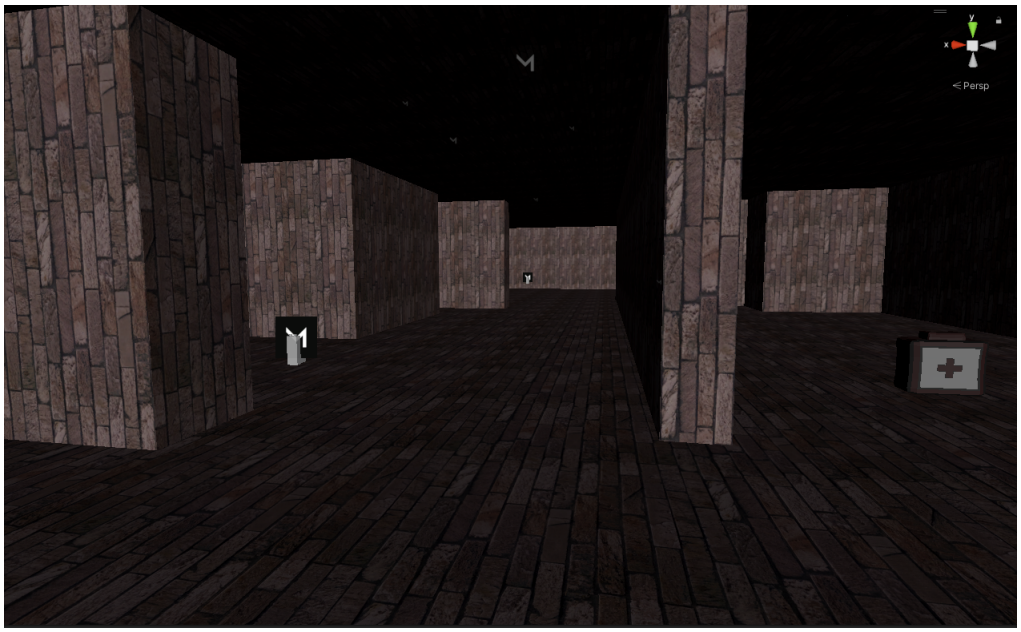


FIGURE 22 – Sous-sol

4.6.2 Probuilder

Pour modéliser la map, nous avons utilisé Probuilder, un outil pré-implémenté dans Unity. Cet outil, particulièrement simple à prendre en main après quelques recherches, nous a permis de bâtir les fondations du manoir ainsi que les différentes pièces le composant. La fonctionnalité "Insert edge loop" a permis de créer les portes et fenêtres, et la fonctionnalité "extrude" nous a servi à élever les murs. Nous avons également pu appliquer les différentes textures et materials grâce à ce dernier.

4.7 Site Web

Pour ce qui est du site web, nous voulons faire quelque chose de simple et assez épuré, nous avons donc décidé de simplement garder le combo html/css/js. Nous utilisions également le préprocesseur *SASS* pour faciliter l'écriture du css. Paul a rejoint Alexis sur le développement du site, il a brillamment su s'adapter notamment en apprenant le *SASS* de manière accélérée.

Le site est réparti en quatre pages, sur chacune il y a une barre de navigation en haut menant aux autres pages ainsi qu'à l'accueil ; ainsi qu'une section pieds de page qui permet de rejoindre notre serveur communautaire Discord et de se rendre sur le dépot Github.

4.7.1 Accueil

Il s'agit de la page principale, on y retrouve une description ainsi qu'un carrousel de captures d'écran du jeu.

4.7.2 Télécharger

La page dédiée au téléchargement du jeu.

4.7.3 À propos

Sur cette page on retrouve des informations sur les membres de notre groupe ainsi que la liste des technologies utilisées pour le projet.

4.7.4 Ressources

C'est ici que l'on peut télécharger tous les documents relatifs au développement du projet.

4.8 Effets sonores

Pour améliorer l'immersion et intensifier l'atmosphère effrayante du jeu, divers bruitages et effets sonores ont été incorporés. Lorsque vous vous déplacez, vous entendrez le son des pas résonner, tandis qu'une sinistre tonalité retentit lorsque l'intelligence artificielle vous détecte. En fermant une porte, vous serez accompagné par le son inquiétant d'un claquement sourd. Enfin, lorsque vous vous trouvez au sous-sol où le gaz est présent, des bruitages de toux viendront ajouter à l'oppression de la situation.

5 Problèmes rencontrés

Durant notre projet, nous avons rencontré plusieurs problèmes. L'un d'entre eux était lié au NavMesh. Lorsque nous avons tenté de placer notre IA dans le manoir, notre clown ne parvenait plus à avancer. Bien que l'animation se déclenchait correctement, il restait immobile. Après un certain temps, nous avons réalisé que le problème résidait dans les paramètres de l'agent du NavMesh ainsi que dans la génération du NavMeshSurface. Heureusement, nous avons rapidement résolu ce problème en ajustant simplement les paramètres de l'agent. Probuilder : Lors de l'application de materials sur certain polygones, cela pouvait le dupliquer pour une raison qui nous échappe encore à l'heure actuelle. Lorsqu'il y avait trop de polygones dupliqués les uns dans les autres, cela déplaçait une partie conséquente de la map lors du lancement du jeu, ce qui le rendait injouable. Il nous a fallu comprendre que certains polygones étaient dupliqués, puis trouver pour supprimer.

6 Ce qui n'a pas été fait

Nous avons globalement bien avancés et sommes tous satisfait de ce que nous avons accomplis. Il reste tout de même de nombreuses choses que nous n'avons pas pu ajouter, que ce soit par manque de temps ou par incapacité à résoudre les problèmes que cela engendrait. Entre autre, nous avons eus de nombreux soucis avec la map qui à tendance à partir en vrille lorsqu'on ajoutait le NavMesh ; l'ajout d'une barre d'endurance pour éviter que le joueur ne fuit indéfiniment n'a pas pu être fait, de même pour la deuxième IA que nous avions prévus de rajouter au sous-sol mais qui posait bien trop de problèmes techniques. En parlant d'IA nous avions d'ailleurs pour premier objectif d'en faire plusieurs avec des spécificités différentes pour chacune d'elles, ce qui n'est pas non plus le cas. Il manque aussi quelques effets sonores (comme lorsqu'on sort ou range la caméra), des interfaces graphiques (pour la victoire ou la défaite) et aussi des accessoires qui auraient permis de ralentir le monstre.

7 Synthèse personelle

7.1 Salim CHARIKH

J'ai principalement consacré mon temps et mes efforts à travailler sur l'intelligence artificielle et les animations dans le projet. J'ai également touché au gameplay, notamment au système de charge de la caméra. En ce qui concerne l'intelligence artificielle, son implémentation a été particulièrement complexe, car c'était un domaine totalement nouveau pour moi. Cependant, malgré les défis rencontrés, cette expérience m'a permis d'apprendre énormément et de m'améliorer considérablement dans ce domaine fascinant.

Quant aux animations, elles ont été plus intuitives et rapides à réaliser. J'ai tout de même rencontré quelques difficultés, mais elles étaient relativement simples à résoudre. Globalement, cette expérience a été incroyablement enrichissante, car elle m'a permis d'acquérir des connaissances approfondies en matière de conception de jeux et de programmation.

7.2 Alexis GALOPIN

Pour ma part j'ai pu toucher un peu à tous les domaines du développement du jeu, mais je suis principalement concentré sur la création des mécaniques de gameplay. De la caméra aux système de portes en passant par l'inventaire ou des choses plus basiques comme les déplacements du joueur, j'ai pu largement faire usage de mes connaissances acquises durant l'année en C#. C'est d'ailleurs sans doute cette partie qui m'a le plus amusée car bien que m'arrachant beaucoup de cheveux à déboguer, j'ai trouvé passionnant le fait de créer ce qui fait le coeur du jeu. Parmi les choses qui

m'ont le plus pris la tête, je citerais en premier les problèmes de NavMesh et de collision avec l'ia ainsi que beaucoup de temps perdu à essayer de régler des conflits git.

7.3 Akomi ZEBILA

Je me suis presque exclusivement chargé de réaliser la map. Cela m'a montré avant tout que créer une carte n'est pas simple. En effet, pour le cas du manoir, il y a eu plusieurs spécificités à prendre en compte : la taille des pièces/couloirs, leur échelle par rapport à la taille du joueur, et la disposition des pièces, fenêtres et escaliers. Cela m'a appris beaucoup de notions dans ce domaine, et j'ai beaucoup aimé passer du temps et utiliser ma créativité pour développer cette map, malgré les quelques problèmes rencontrés.

7.4 Paul MARIONNET

Pour ce qui est de ma participation, je me suis chargé des menus du jeu ainsi que du réseau et de la mise en place d'un système de lobby. Ces parties étant presque indépendante des autres, mon adaptation suite à mon transfert dans ce groupe a été grandement optimisée et le temps qu'il m'a fallu pour comprendre la structure préexistante en a été également extrêmement diminué. La partie qui m'a le plus interessé était sans aucun doute la partie UI. Ma prise en main des outils était fluide et la création naturelle. J'ai eu quelques difficultés principalement sur les lobbys qui ont nécessité de nombreuses heures de debugging pour fixer tous les problèmes de gestion de lobby ainsi que d'optimiser les requêtes envoyées à l'API.

8 Conclusion

L'équipe AVAC# est fière d'annoncer que notre projet Stuck est enfin terminé, après plusieurs mois de travail acharné. Au cours de ce processus, nous avons tous acquis de nouvelles compétences, que ce soit dans la création de musique, la modélisation de cartes, la programmation en C, l'utilisation du logiciel Unity et bien d'autres aspects encore.

Malgré les difficultés rencontrées pendant le développement, nous avons su trouver des solutions adaptées pour les surmonter. Nous avons pris plaisir à développer de nouvelles fonctionnalités et à laisser libre cours à notre imagination pour créer un univers unique autour de notre projet.

Bien sûr, notre projet peut toujours être amélioré et il comporte certains défauts, mais nous sommes fiers d'annoncer que nous avons atteint tous les objectifs principaux définis dans notre cahier des charges.

Nous souhaitons exprimer notre gratitude aux membres du jury pour leur attention tout au long de ce projet.

- AVAC#

Table des figures