

Chapitre 1 : État de l'art

1.1 MOTEURS DE JEU

Pour appréhender le processus de création d'un jeu vidéo, il convient de se questionner sur les méthodes et outils utilisés dans sa conception. La création d'un jeu vidéo repose en effet sur un ensemble d'étapes techniques et créatives, dont l'élément central est le moteur de jeu.

Mais qu'est-ce qu'un moteur de jeu ? Un moteur de jeu est un ensemble d'outils et de technologies qui permet de créer des jeux vidéo en intégrant plusieurs fonctionnalités comme la gestion de la physique, le rendu graphique, l'animation, le son, le scripting, l'interface utilisateur, et bien d'autres. C'est grâce à ce moteur que les développeurs peuvent construire, programmer et tester leur jeu dans un environnement cohérent et interactif.

1.2 MÉTHODES RIGOUREUSES DANS L'INDUSTRIE DU JEU VIDÉO

L'industrie du jeu vidéo adopte des méthodologies rigoureuses pour garantir la qualité et l'excellence des jeux. Voici quelques-unes des approches les plus couramment utilisées :

1.2.1 Méthodes de brainstorming

Les équipes de développement de jeux utilisent des sessions de brainstorming pour générer des idées novatrices. Ces sessions impliquent souvent des membres de diverses disciplines, y compris des concepteurs de jeux, des artistes, des programmeurs et des spécialistes du marketing. Cela permet d'obtenir une variété de perspectives et d'expertises.

Techniques de brainstorming :

- **Mind Mapping** : Les équipes utilisent des diagrammes pour visualiser les idées et les relier entre elles, ce qui aide à identifier des concepts potentiels pour le jeu.
- **Brainwriting** : Chaque participant écrit ses idées sur un papier, puis passe le papier à un autre membre de l'équipe, qui peut ajouter des commentaires ou développer l'idée, favorisant ainsi l'émergence de nouvelles idées.

1.2.2 Intégration du Design Thinking

Le design thinking est une approche centrée sur l'utilisateur qui vise à comprendre les besoins et les désirs des joueurs. Il se compose généralement des étapes suivantes : empathie, définition, idéation, prototypage et test.

- **Exemple :** Lors du développement d'un jeu d'aventure, une équipe peut commencer par des entretiens avec des joueurs pour comprendre leurs préférences en matière de gameplay, de narration et d'interaction. Ensuite, l'équipe définit les principaux défis que le jeu doit surmonter, avant de passer à l'idéation et au prototypage.

1.2.3 Moteurs de Jeu Personnalisés

De nombreuses sociétés de jeux vidéo optent pour le développement de moteurs de jeu sur mesure afin de répondre à des exigences spécifiques de leurs projets. Ce choix leur confère un contrôle total sur les fonctionnalités, l'optimisation et la performance des jeux, tout en intégrant des outils adaptés à leurs styles de gameplay distinctifs.

Activision, par exemple, utilise le Treyarch Engine pour la série Call of Duty, un moteur qui permet d'améliorer en continu les graphismes et les mécanismes de jeu, tout en offrant une expérience multijoueur fluide.

De son côté, **Rockstar Games** a développé le RAGE (**Rockstar Advanced Game Engine**), employé dans des titres emblématiques comme Grand Theft Auto V et Red Dead Redemption 2. Ce moteur est reconnu pour sa gestion de mondes ouverts vastes et détaillés, ainsi que pour ses systèmes d'intelligence artificielle avancés.

Ubisoft a conçu le moteur Anvil, qui soutient la série Assassin's Creed. Anvil est particulièrement efficace dans la gestion des animations et des environnements historiques, ce qui contribue à l'immersion des joueurs.

Electronic Arts (EA), quant à elle, exploite le Frostbite Engine, un moteur qui se distingue par sa capacité à produire des graphismes réalistes et des environnements destructibles, enrichissant ainsi l'expérience de jeu dans des franchises comme Battlefield et FIFA.

En outre, **Epic Games** utilise l'Unreal Engine, un moteur également adopté par de nombreuses autres sociétés en raison de sa flexibilité et de ses performances exceptionnelles. L'Unreal Engine est apprécié pour ses outils avancés, sa compatibilité avec la réalité virtuelle et augmentée, et sa communauté dynamique qui partage des ressources et des plugins.

Unity Technologies, avec son moteur Unity, constitue également une plateforme de choix pour le développement de jeux 2D et 3D. Ce moteur est souvent privilégié pour sa facilité d'utilisation et son écosystème riche, permettant de cibler plusieurs plateformes, y compris les appareils mobiles et VR.

La création de moteurs de jeu sur mesure présente l'avantage d'assurer des mises à jour et des améliorations adaptées aux spécificités des projets. Les équipes de développement peuvent concevoir des outils personnalisés pour les processus de création de contenu, tels que la modélisation 3D, l'animation et le rendu graphique, ce qui optimise l'efficacité et la qualité globale des jeux. Cette stratégie favorise également la différenciation sur un marché compétitif en exploitant pleinement les capacités technologiques de leurs moteurs.

1.2.4 Modèle 3D

La modélisation 3D dans l'industrie du jeu vidéo repose sur divers logiciels et matériels. Des outils tels que **Blender**, **Autodesk Maya**, et **3ds Max** sont largement utilisés pour créer des modèles 3D, offrant des fonctionnalités avancées pour la modélisation, le texturing et l'animation. Le photoréalisme est souvent atteint grâce à des techniques comme le **baking des textures** et l'utilisation de shaders complexes.

Pour la création de modèles, des tablettes graphiques, comme celles de **Wacom**, permettent aux artistes de sculpter et de peindre numériquement avec précision. Les scanners 3D sont également utilisés pour capturer des objets réels, fournissant des données précises qui peuvent être intégrées dans les modèles virtuels.

Les techniques de texture, telles que le **PBR (Physically Based Rendering)**, sont essentielles pour simuler des matériaux réalistes. Les logiciels de rendu, tels que **V-Ray** et **Arnold**, sont employés pour produire des images réalistes, augmentant l'immersion visuelle des jeux. L'ensemble de ces outils et techniques contribue à créer des environnements 3D riches et dynamiques.

1.2.5 Optimisation

Les **majors de l'industrie du jeu vidéo**, telles qu'Activision, Ubisoft, Electronic Arts et Rockstar Games, emploient diverses techniques pour optimiser les graphismes et les performances de leurs jeux. Elles utilisent le **Level of Detail (LOD)**, qui permet de varier les modèles 3D selon la distance du joueur, réduisant ainsi la charge graphique. Le **culling** est également essentiel, car il élimine les objets non visibles pour la caméra, comme ceux se trouvant derrière le joueur, ce qui diminue le nombre d'objets à dessiner. Pour améliorer les performances, ces studios recourent au **baking des lumières**, qui pré-calcule les effets d'éclairage dans les textures, limitant ainsi le besoin de calculs en temps réel. De plus, elles développent des **shaders optimisés** qui consomment moins de ressources tout en maintenant un bon rendu visuel.

En ce qui concerne la gestion de la mémoire, ces studios utilisent des outils de **profiling** pour surveiller l'utilisation de la mémoire et détecter les fuites ou surcharges. Ils appliquent également le **streaming de contenu**, qui charge dynamiquement les ressources comme les textures et les modèles en fonction de la position du joueur, évitant ainsi un chargement initial trop lourd.

L'optimisation du code est une priorité pour ces entreprises. Elles pratiquent le refactoring pour améliorer l'efficacité du code, réduire sa complexité et en augmenter la lisibilité. L'application de techniques de **programmation orientée objet** permet également de structurer le code de manière plus efficace et réutilisable, tandis que le choix des **algorithmes les plus efficaces** pour des tâches courantes, comme l'intelligence artificielle, est une pratique courante.

Les tests et retours sont intégrés tout au long du processus de développement. Le **beta testing** est une méthode fréquemment utilisée pour recueillir des retours sur les performances et corriger les problèmes avant la sortie. Les **majors de l'industrie** écoutent également le feedback **des joueurs** pour identifier rapidement les problèmes de performance et les résoudre.

Pour le développement multi-plateformes, elles créent des jeux **cross-platform** qui s'adaptent automatiquement aux spécificités de chaque plateforme, optimisant ainsi les performances. Cela inclut aussi la **réduction des exigences matérielles** en proposant des versions « légères » des jeux pour des consoles et PC moins puissants.

Ces entreprises publient régulièrement des **patchs et mises à jour** post-lancement pour corriger les bugs et améliorer les performances en réponse aux problèmes signalés par les joueurs.

1.2.6 Narration

La narration constitue un élément clé des jeux vidéo modernes, enrichissant l'expérience des joueurs par des récits captivants. Des scénaristes professionnels et des équipes dédiées s'engagent dans la création d'histoires engageantes, comme le démontre The Last of Us, où l'histoire et le gameplay se complètent harmonieusement.

Les récits peuvent être linéaires ou non linéaires, comme dans The Witcher 3, où les choix des joueurs influencent l'évolution de l'intrigue et des personnages. Cette dynamique nécessite une planification soignée pour que chaque décision ait des conséquences significatives, renforçant ainsi l'engagement émotionnel.

L'intégration de la narration dans le gameplay se manifeste par des quêtes secondaires, des dialogues interactifs et des événements influencés par les actions des joueurs. La technologie, notamment la capture de mouvement et l'enregistrement audio de haute qualité, contribue à la création de récits immersifs.

La collaboration interdisciplinaire entre les équipes de conception, de programmation, d'art et de son est essentielle pour assurer une intégration fluide de la narration dans l'expérience globale du jeu.

1.2.7 Audio

L'industrie du jeu vidéo utilise des techniques audio avancées pour renforcer l'immersion des joueurs. Des systèmes audios spatialisés, gérés par des logiciels comme **FMOD** et **Wwise**, permettent de créer un son directionnel, rendant l'expérience plus réaliste.

Les enregistrements sont souvent réalisés dans des studios équipés de microphones de haute qualité, tels que le **Neumann U87**, et d'interfaces audio performantes. Les effets sonores générés par ordinateur sont créés avec des logiciels comme **Adobe Audition**, **Logic Pro** ou **Ableton Live**, ajoutant profondeur et ambiance sonore.

La **capture de mouvement** synchronise les performances vocales avec les animations des personnages, tandis que des systèmes de son surround comme **Dolby Atmos** offrent une expérience audio tridimensionnelle. Ces techniques et matériels assurent une atmosphère sonore immersive et dynamique dans les jeux vidéo.

1.2.8 Création et gestion des animations

La capture de mouvement (Motion Capture) est une méthode clé pour créer des animations réalistes. Des acteurs sont équipés de capteurs qui enregistrent leurs mouvements, lesquels sont transférés à des modèles 3D. Cette technique permet d'obtenir des animations fluides et naturelles, rendant le gameplay plus engageant.

Des systèmes de capture, tels que les caméras infrarouges et les costumes dotés de marqueurs, sont utilisés pour suivre les mouvements avec précision. Les données recueillies sont traitées par des logiciels comme MotionBuilder ou Maya, qui facilitent l'intégration des mouvements capturés dans les jeux.

Cette approche améliore non seulement la qualité des animations, mais contribue également à créer des interactions plus réalistes entre les personnages et leur environnement. En

somme, la capture de mouvement est essentielle pour donner vie aux personnages dans les jeux vidéo.

1.2.9 Interface

L'interface utilisateur (UI) et l'expérience utilisateur (UX) sont essentielles dans la conception des jeux vidéo, influençant l'interaction des joueurs avec le gameplay. Des outils comme Adobe XD, Sketch et Figma sont utilisés pour créer des prototypes d'interfaces intuitives, tandis que des logiciels tels que Photoshop et Illustrator servent à concevoir les éléments visuels, comme les menus et les HUD.

La recherche utilisateur, par le biais de tests de jeu, permet de recueillir des retours pour affiner l'interface et optimiser l'expérience. Les principes de design, tels que la hiérarchie visuelle et la cohérence, guident les joueurs et renforcent l'immersion. L'adaptabilité des interfaces aux différentes plateformes est également cruciale pour garantir une expérience fluide.

1.3 CONFIGURATION MATÉRIELLE

Le développement de ce projet a été réalisé sur un ordinateur Acer TravelMate P245-MG, sortie en 2014, équipé d'un processeur Intel Core i5 et de 16 Go de RAM. Cette configuration à double carte graphique comprend une Intel HD Graphics et une NVIDIA GeForce 820M.

1.4 CHOIX DES LOGICIELS

Dans le cadre du développement de ce jeu, divers logiciels ont été envisagés, chacun apportant des fonctionnalités spécifiques adaptées aux besoins du projet.

1.4.1 Unity3D

Le choix d'Unity3D (C#) s'explique par sa simplicité, sa vaste communauté et ses nombreuses ressources, idéales pour les projets indépendants. Bien que Godot (C#) soit gratuit et opensource, il est limité pour des fonctionnalités avancées comme les animations complexes. Unreal Engine (C++), bien que puissant pour le rendu photoréaliste, a une courbe d'apprentissage plus abrupte, ce qui le rend moins adapté aux petites équipes. La version LTS d'Unity 2022 a été retenue pour sa stabilité et son support à long terme (voir Annexe 1).

1.4.2 Choix de Langage : C# ou Visual Scripting

Dans ce projet, la programmation en C# a été choisie malgré les avantages du Visual Scripting, une méthode visuelle basée sur le "drag-and-drop" de blocs de commandes. Bien que plus accessible, Visual Scripting est moins performant et précis que C#, essentiel pour répondre aux besoins du jeu. Les raisons incluent :

- **Contrôle et Flexibilité** : C# offre un contrôle granulaire sur la logique du jeu, permettant des optimisations précises.
- **Performance** : Le code C# peut être plus performant, crucial dans des projets complexes.
- **Ressources et Documentation** : Une communauté large et une documentation abondante facilitent la recherche de solutions.
- **Réutilisabilité et Évolutivité** : C# permet de développer des modules réutilisables, favorisant l'évolutivité pour des projets futurs et simplifiant la maintenance du code.
- **Expérience de Développement** : La familiarité avec C# maximise l'exploitation des fonctionnalités avancées de Unity.

1.4.3 Choix d'Intelligence Artificielle : IA Traditionnelle ou Machine Learning

L'IA traditionnelle a été privilégiée en raison de sa rapidité de mise en œuvre et de son contrôle précis sur les comportements. Contrairement au Machine Learning, qui nécessite des phases d'entraînement prolongées et un matériel performant (notamment avec TensorFlow ou PyTorch), le codage traditionnel permet de définir manuellement des comportements et de restreindre facilement les actions de l'IA. Cette méthode est ainsi plus rapide à déboguer et mieux adaptée aux arbres de décision simples, tout en répondant aux contraintes matérielles du projet.

1.4.4 Blender

Blender s'est imposé pour plusieurs raisons :

Gratuité et accessibilité : Contrairement à Maya et 3DS Max, Blender est entièrement gratuit, un atout majeur pour les développeurs indépendants.

Polyvalence et efficacité : Ce logiciel intègre en une seule interface des outils de modélisation, d'animation, de texturage et de rendu, offrant un workflow fluide, notamment pour l'exportation d'assets vers Unity.

Intégration avec Unity : Blender permet une exportation directe des fichiers au format FBX (Filmbox), garantissant ainsi une bonne compatibilité avec Unity3D.

1.4.5 Mixamo

Mixamo a été utilisé pour simplifier l'intégration des animations de personnages, qui demandent beaucoup de temps. Ce site offre une bibliothèque d'animations et de personnages

prêts à l'emploi, ce qui facilite le processus d'animation et permet un gain de temps significatif dans la mise en place des mouvements.

1.4.6 FL Studio

L'audio constitue un élément essentiel pour l'immersion dans les jeux vidéo. Le choix de FL Studio pour la création et l'édition des musiques et effets sonores a été motivé par sa flexibilité et son interface intuitive. Ce logiciel offre une précision élevée pour le mixage et la création de boucles sonores, permettant d'intégrer des ambiances harmonieuses et adaptées aux différentes scènes du jeu.

1.4.7 Photoshop

Adobe Photoshop a été utilisé pour concevoir l'interface utilisateur (UI) du jeu grâce à ses fonctionnalités avancées, permettant de créer rapidement des éléments visuels comme boutons et icônes. Les fichiers sont facilement exportables en sprites pour Unity.

1.4.8 Lightroom

Adobe Lightroom a été utilisé pour optimiser les images, en ajustant la colorimétrie et les contrastes pour assurer une cohérence esthétique des assets graphiques du jeu.

1.4.9 Premiere Pro et After Effects

Pour le montage vidéo et les cinématiques, Adobe Premiere Pro a été utilisé pour le montage, tandis qu'After Effects a permis de créer des effets visuels avancés, renforçant l'identité visuelle du jeu.

1.4.10 Sources et sites de téléchargement de ressources

Plusieurs sites web, tels que Pixabay et CG Ambiance, ont fourni des ressources libres de droits, comme des textures **PBR** et des fichiers audios, qui ont ensuite été adaptées et intégrées dans le jeu.

Chapitre 2 : Conception du jeu “AIZA”

2.1. CONCEPT DU JEU

Le jeu “AIZA” est un jeu d'aventure qui plonge le joueur dans un univers apocalyptique. Le personnage principal, une version fictive d'un individu malgache, évolue dans un monde ravagé, principalement situé à Madagascar, et se concentre sur le lieu spécifique de MATERAUTO à Ankorondrano. Le public ciblé est principalement composé de joueurs malgaches.

2.2. INSPIRATIONS ET MÉCANIQUES DE JEU

Le gameplay s'inspire de divers titres renommés, intégrant des éléments de Last of Us pour la narration et l'interaction, ainsi que des mécaniques de GTA et Call of Duty pour l'exploration et le combat. Le monde du jeu se réfère également à l'environnement immersif de Red Dead Redemption, créant ainsi une expérience de jeu riche et variée.

2.3. CHOIX ARTISTIQUE ET TECHNIQUE

Le style artistique du jeu est fortement ancré dans la culture malgache, utilisant des éléments sonores et visuels qui évoquent l'ambiance locale. La musique intégrée, principalement triste, contribue à l'atmosphère générale du jeu et renforce l'immersion du joueur. Les éléments graphiques sont réalistes et en 3D, visant à créer un environnement de jeu convaincant.

2.4. MÉCANISMES DE JEU

Les mécanismes de jeu permettent au joueur de se déplacer librement dans le monde, d'interagir avec divers objets, et d'engager des dialogues avec des personnages non-joueurs. Le système de combat est intégré, offrant des possibilités de gestion des ressources et d'objets. La mort du personnage peut survenir dans certaines situations, ajoutant un élément de défi au gameplay.

2.5. ÉLÉMENTS D'INTERACTION

Le script de gestion des interactions a été écrit en **C#** dans **Unity**. Cela inclut la gestion des animations, des mécaniques de déplacement, et de l'intelligence artificielle des ennemis, avec le NavMesh pour gérer les déplacements et les comportements stratégiques des adversaires. Des arbres de décision ont été implémentés pour rendre l'IA plus dynamique et imprévisible.

Chapitre 3 : Réalisation du jeu

3.1 MODÉLISATION 3D AVEC BLENDER

3.1.1 Modélisation du personnage contrôlable

La création a débuté par la modélisation détaillée de la tête en haute résolution via l'outil "Sculpt" de Blender, optimisée ensuite par décimation et retopologie avec l'add-on B-Surface. Des photos de référence ont guidé les ajustements de proportions. La modélisation du corps a été réalisée par moitiés avec le modificateur "Mirror" et adoucie via "Smooth".

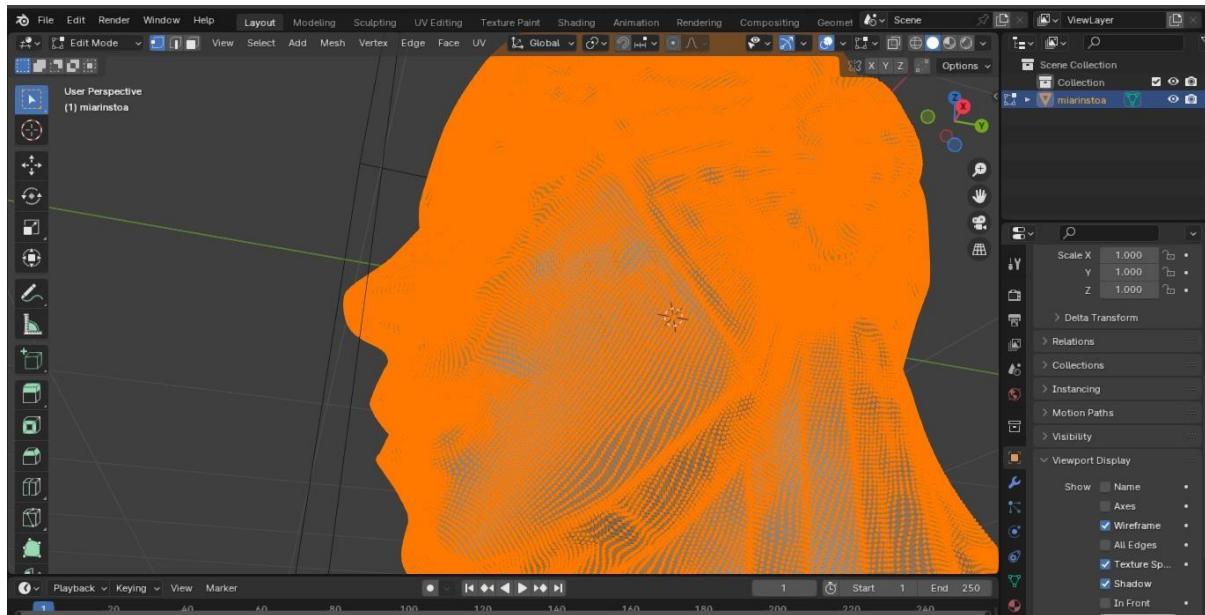


Figure (3, 1) : Résultat du sculpte

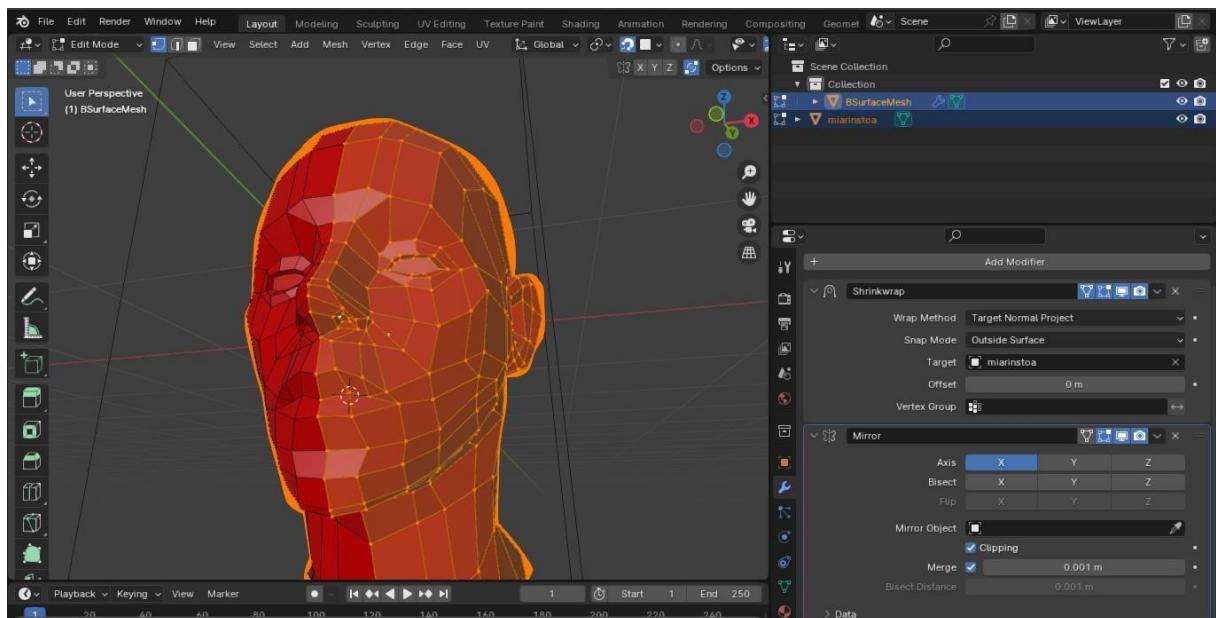


Figure (3, 2): Résultat retopologie

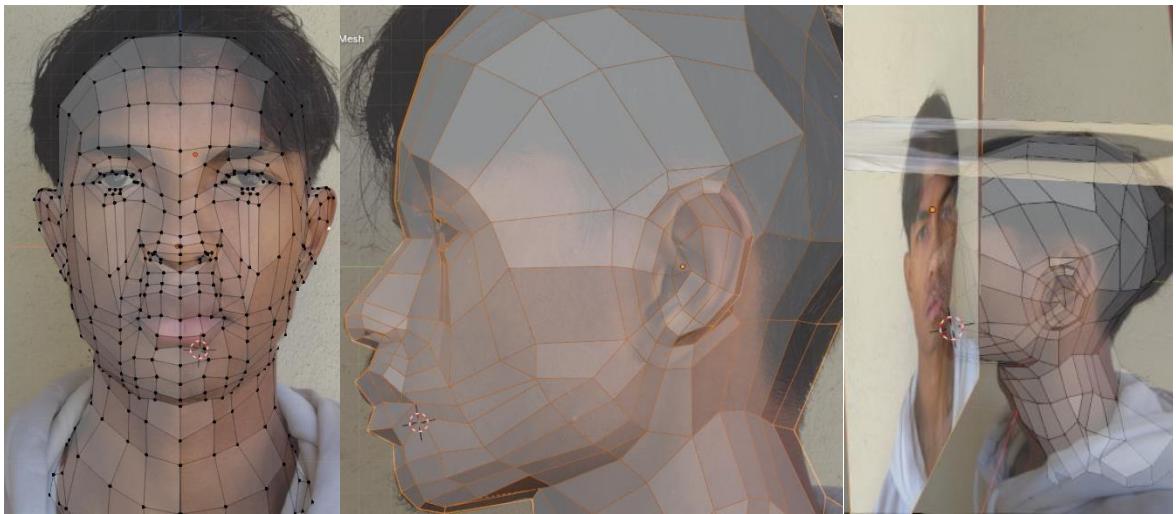


Figure (3, 3) : Images de références

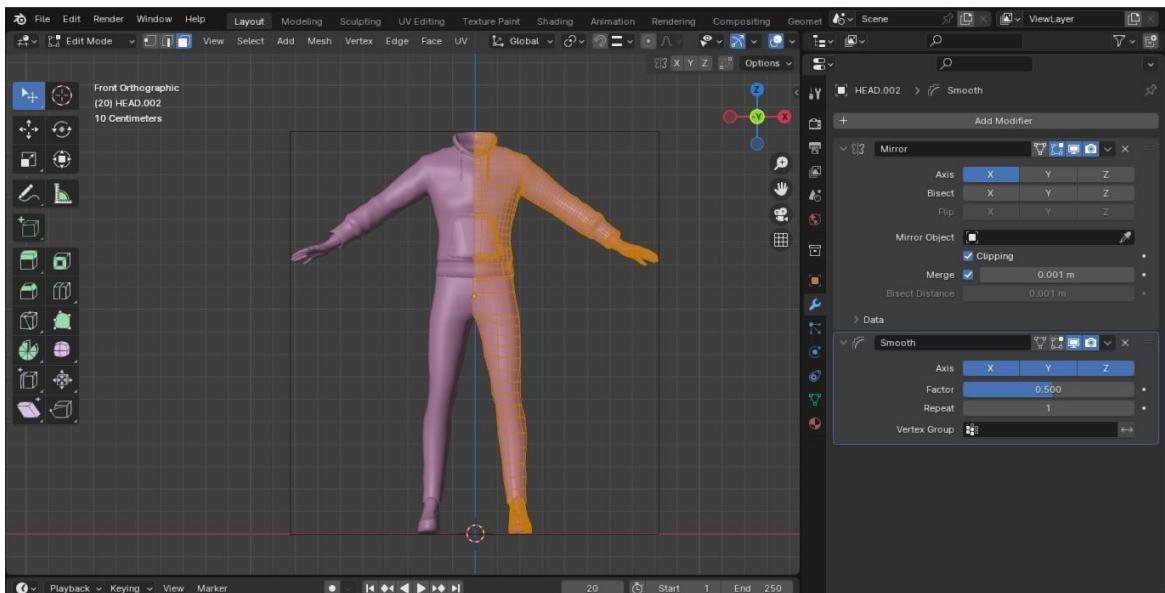


Figure (3, 4) : Modélisation semi-corps

3.1.2 Texturisation

Pour la texturisation, des coutures (Mark Seam) ont été marquées pour faciliter le dépliage UV du visage, puis des textures réalistes ont été appliquées via la projection UV Mapping. Les erreurs d'alignement et d'artefacts ont été corrigées dans Photoshop. Corps, masque et lunette ont été texturés séparément. Les parties texturées ont été fusionnées en un seul maillage, mais chaque élément a conservé sa propre texture et UV Map. Les Nodes de Blender ont servi à affiner les textures (BaseColor, NormalMap, Metallic), avec un baking final pour la compatibilité Unity.

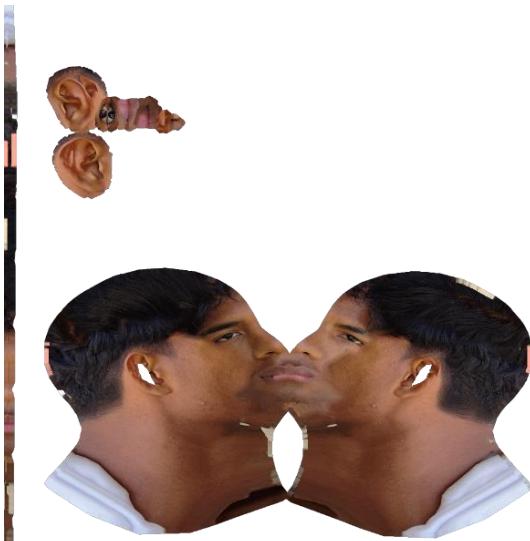


Figure (3, 5) : Texture après rendu

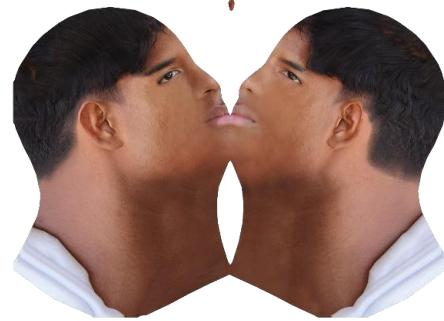


Figure (3, 6) : Texture après correction

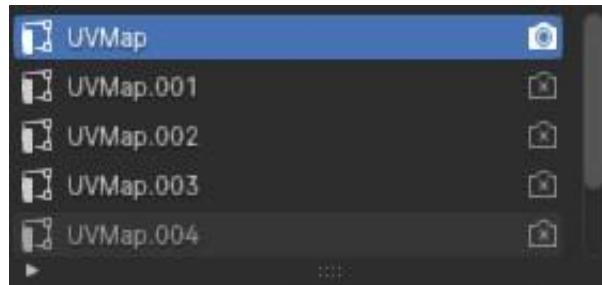


Figure (3, 7) : UVMaps de l'objet fusionné

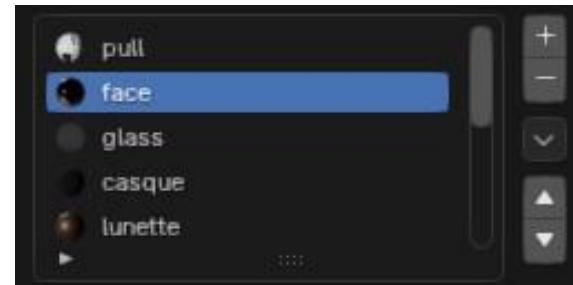


Figure (3, 8) : Slots de l'objet fusionné

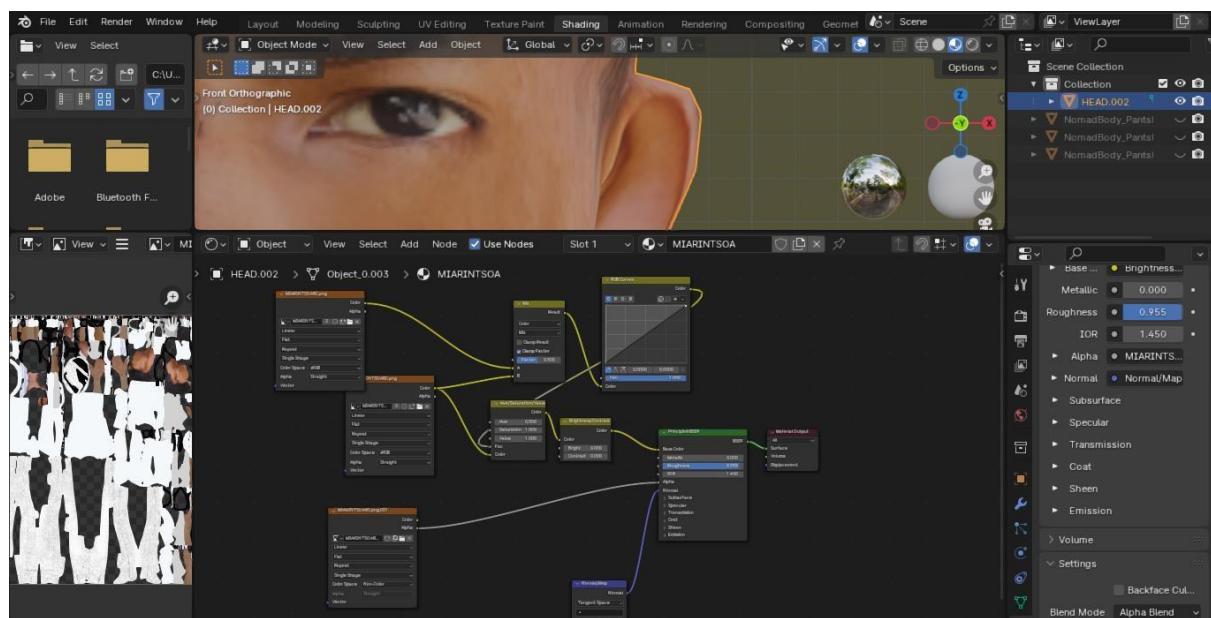


Figure (3, 9) : Shader Nodes



Figure (3, 10) : Résultat personnage

3.1.3 Animation

Pour les animations, l'add-on Rigify de Blender a été utilisé, en particulier le metarig Human, qui facilite la création d'une armature complète. Le metarig, une structure initiale simplifiée, a permis de générer un rig final avec les contrôles nécessaires pour l'animation. Certaines bones(os) inutiles ont été supprimées, et les restantes repositionnées pour correspondre au personnage. Pour associer le metarig au modèle, l'option "Empty Groups" a été utilisée, permettant une affectation manuelle des influences avec le mode "Weight Paint". Cette méthode a évité les erreurs de déformation et d'influences non désirées souvent causées par l'option "Automatic Weights".

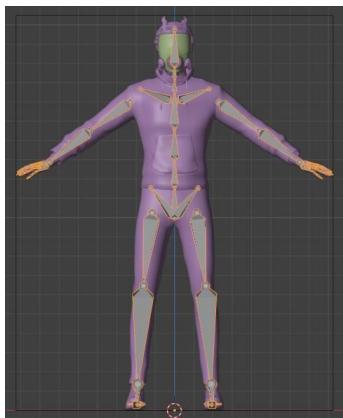


Figure (3, 11) : Armature ajusté

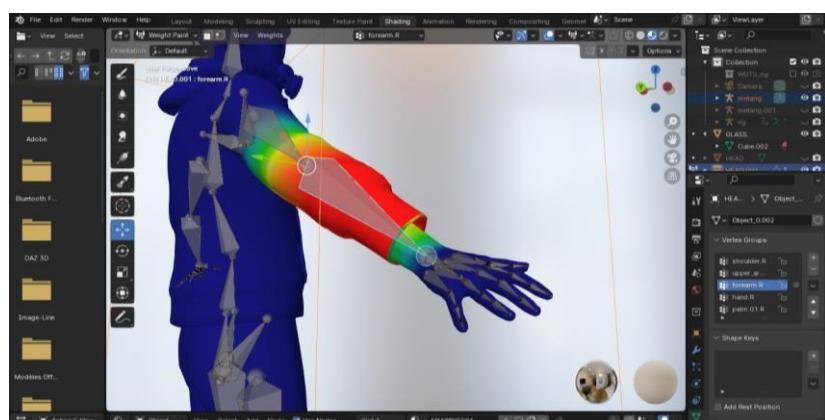


Figure (3, 12) : Ajustement manuel des influences des os

Les animations ont été réalisée en Pose Mode, en utilisant plusieurs fenêtres essentielles :

- ❖ **Dope Sheet** : pour visualiser et organiser les actions d'animation.
- ⌚ **Timeline** : pour positionner et ajuster les images clés.
- ⚡ **Graph Editor** : pour affiner les courbes d'animation et assurer la fluidité des mouvements.

Les animations, conçues pour boucler, partagent les mêmes paramètres au premier et dernier keyframe, garantissant une transition fluide et naturelle.

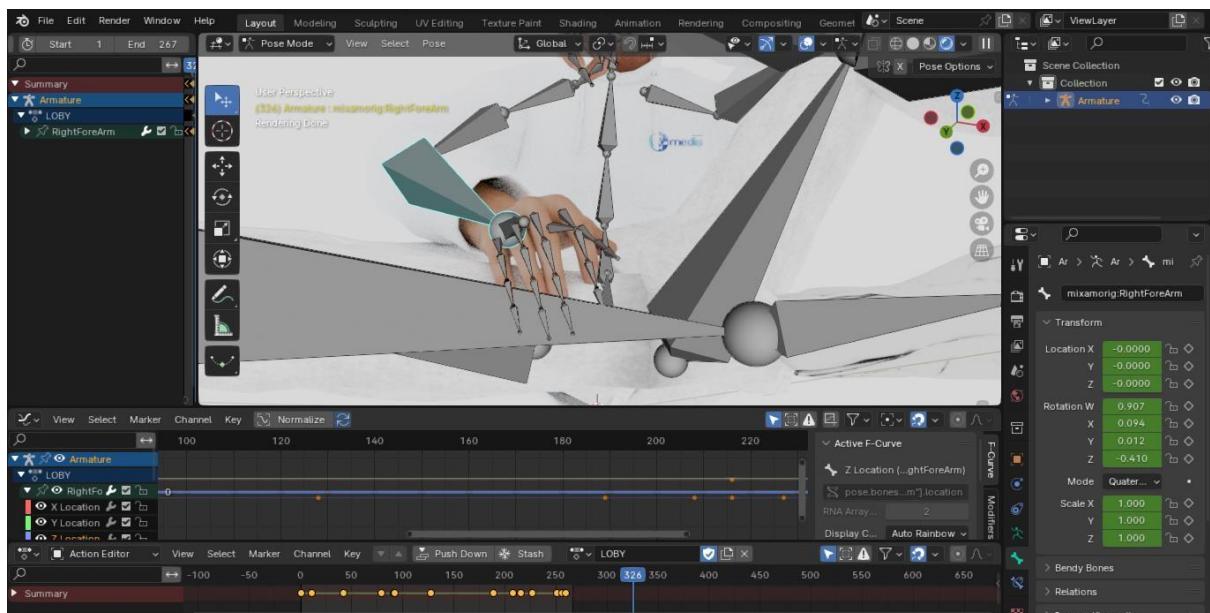


Figure (3, 13) : Gestion des animations dans Blender

Pour renforcer le réalisme, des imperfections subtiles ont été ajoutées aux mouvements. Le modificateur Noise dans le Graph Editor a introduit des variations aléatoires, évitant ainsi un effet artificiel dû à des animations trop uniformes.

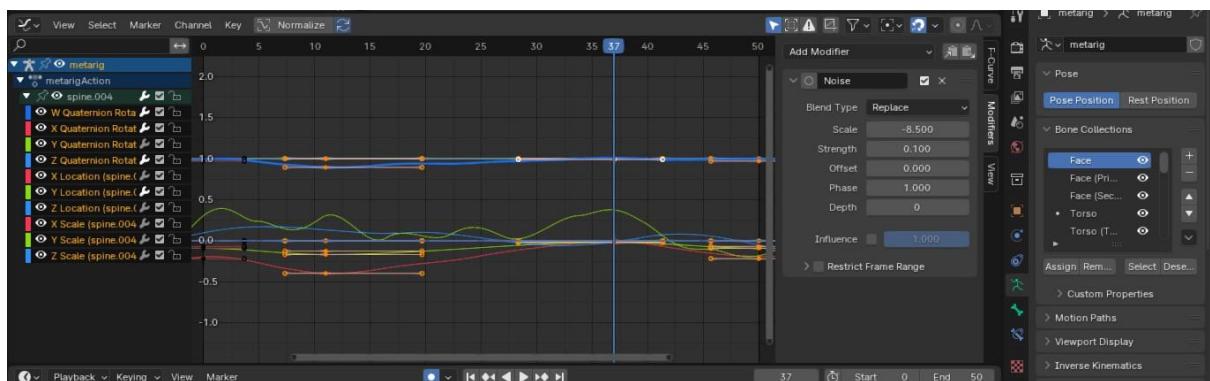


Figure (3, 14) : Courbes d'animation dans Graph Editor

3.1.4 Modélisation environnement MATERAUTO/Ankorondrano

La modélisation 3D de l'environnement a commencé par l'importation d'images de référence, notamment de Google Maps, pour garantir une échelle précise. Chaque élément a été modélisé avec soin en utilisant des techniques de texturage et de baking similaires à celles des personnages, assurant ainsi une cohérence visuelle.

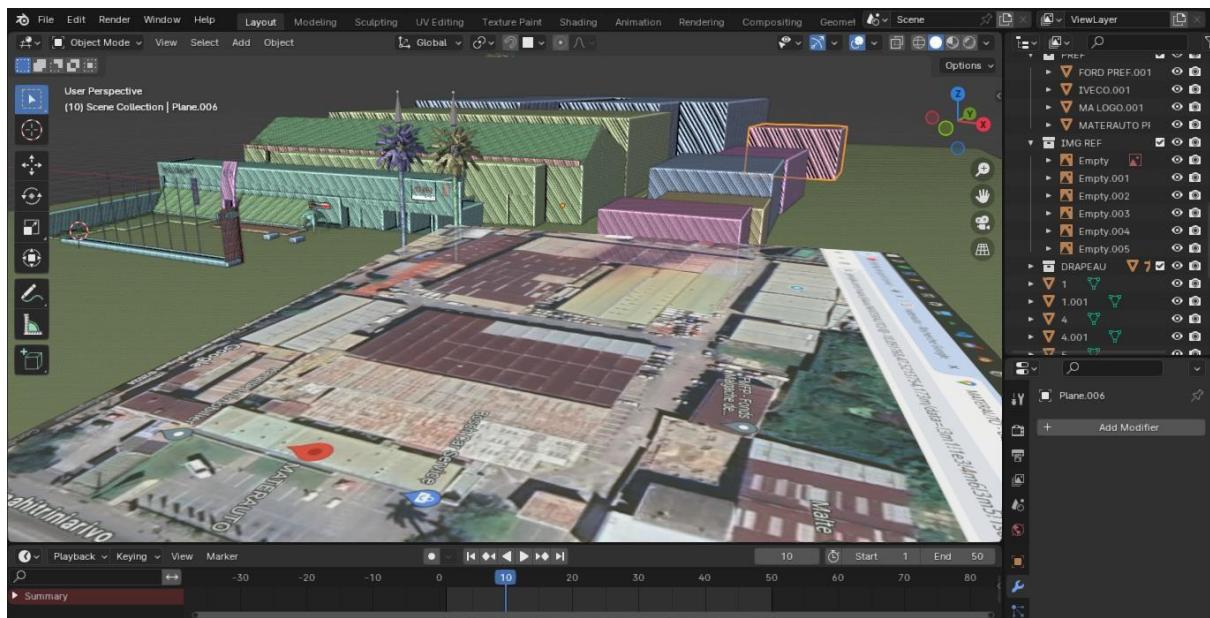


Figure (3, 15) : Modelisation de l'environnement “Materauto”

Pour optimiser le projet, certains détails ont été simplifiés ou omis, remplaçant les éléments complexes par des objets simplifiés, tout en préservant l'esthétique apocalyptique et l'intégrité visuelle.

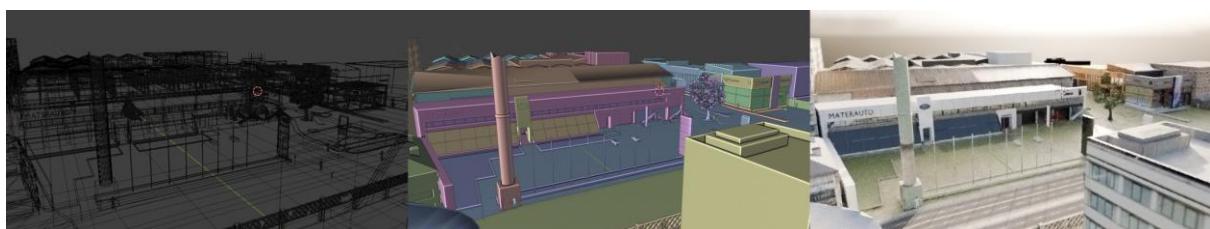


Figure (3, 16) : Aperçu de l'environnement dans blender sous plusieurs rendus

3.1.5 Exportation vers Unity

Pour l'exportation vers Unity, les formats supportés incluent FBX, OBJ, GLTF et Collada, le format FBX étant privilégié pour sa prise en charge des animations, textures et matériaux. Les paramètres d'exportation ont été réglés sur Z Forward et Y Up pour adapter les différences d'axes entre Blender et Unity.

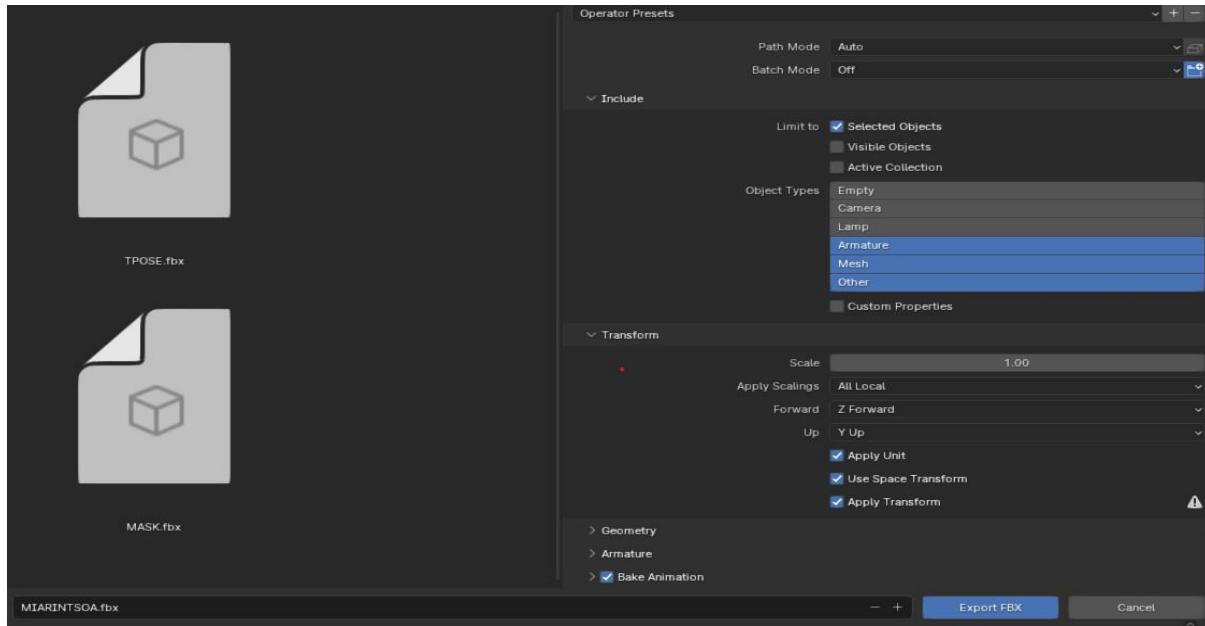


Figure (3, 17) : Exportation en FBX

Il est crucial d'appliquer toutes les transformations dans Blender pour garantir que les objets conservent des transformations cohérentes lors de leur importation et des modifications. Les options Apply Transform et Bake Animation ont été activées pour assurer l'exportation correcte des animations.

3.2 MONTAGE VIDÉO DANS PREMIERE PRO ET AFTER EFFECTS

La narration est cruciale dans les jeux vidéo pour immerger les joueurs. Pour ce projet, une histoire fictive inspirée de fait réel Madagascar a été développée.

Quatre vidéos ont été créées pour enrichir cette narration, utilisant des enregistrements d'écran, des contenus téléchargés et des rendus Blender. Elles ont été exportées au format MP4, à 30 FPS et en résolution 1366x768 pixels pour une intégration de qualité.

3.2.1 Vidéo Logo

Création Logo MITRIINI dans Blender

Le logo est une animation 3D de quatre cubes inspirés par Unity 3D, modélisés dans Blender. Un matériau combinant Glass BSDF et Shader Emission crée des reflets irisés, tandis qu'un Shader Volume Scatter ajoute des nuances colorées. L'effet Bloom génère une lueur, et la caméra en mode orthographique capte l'esthétique. Le moteur de rendu Cycles est utilisé pour des rendus réalistes.

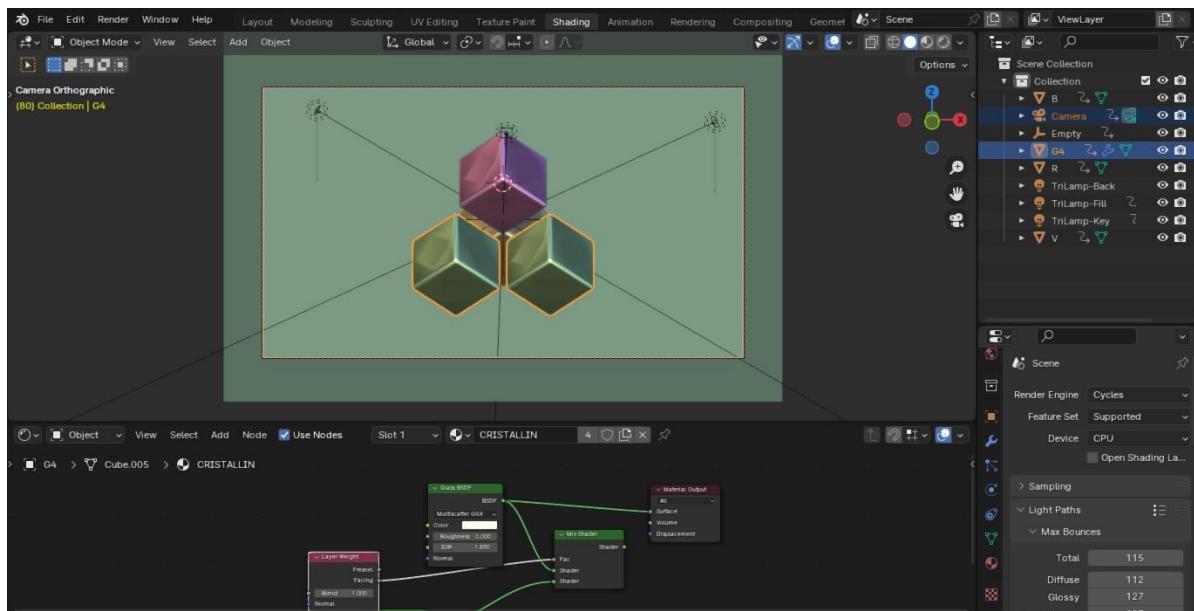


Figure (3, 18) : Modélisation et animation du logo dans Blender

Montage dans After Effects

Les textes, logos et images ont été vectorisés et retravaillés dans Photoshop, puis importés dans Adobe After Effects pour créer des représentations d'autres logos et divers effets spéciaux.



Figure (3, 19) : Extrait du montage de l'animation dans After Effects

3.2.2 Vidéo de la capture d'écran

La vidéo a été capturée avec un téléphone et monté dans Premiere Pro, en supprimant les séquences inutiles. L'audio a été nettoyé et amélioré dans FL Studio, et des effets sonores supplémentaires, comme des tremblements et des explosions, ont été ajoutés.

3.2.3 Vidéo générique

Le montage s'est déroulé en deux étapes, utilisant deux logiciels pour garantir un résultat optimal.

Création du générique dans Adobe Premiere Pro

Premiere Pro a servi à découper la vidéo, organiser les pistes audio et structurer les clips dans la timeline. Des ajustements de vitesse ont été effectués, et des effets visuels comme Lumetri Color, Gaussian Blur, Warp Stabilizer et DeNoise ont été appliqués pour améliorer la qualité. L'audio a été synchronisé avec les vidéos, intégrant des effets sonores tels que des explosions et des extraits de journaux télévisés.

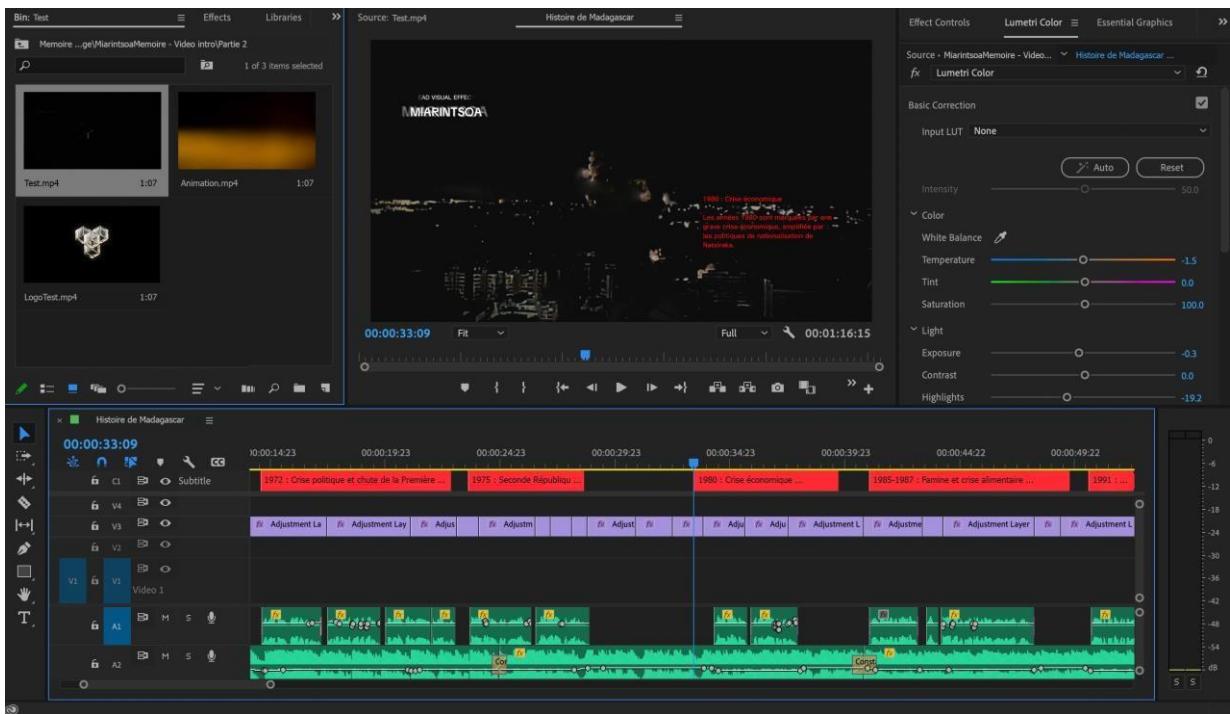


Figure (3, 20) : Montage du générique dans Adobe Premiere Pro

Montage dans Adobe After Effect

After Effects a été utilisé pour des superpositions vidéo, appliquant des modes de fusion comme Overlay et Soft Light pour enrichir les effets visuels. Des textes ont été animés avec des effets tels que Fade in/out, Wiggle, Typewriter et Blur.

3.2.4 Vidéo de fin sur l'histoire du personnage principal

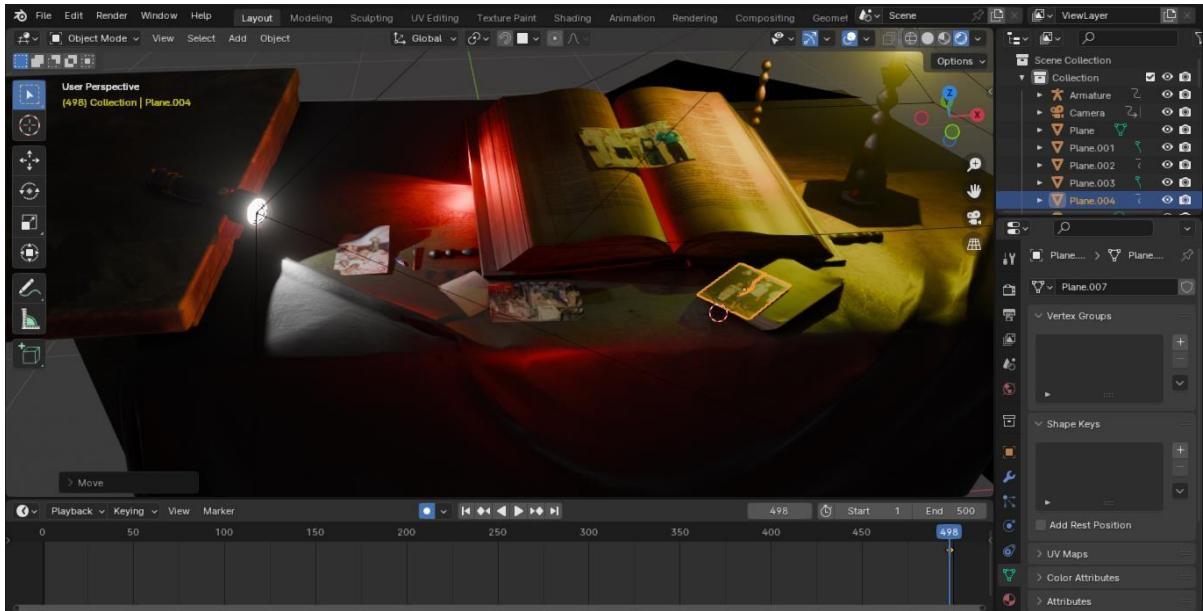


Figure (3, 21) : Montage de fin d'histoire dans Blender

3.3 CRÉATION D'INTERFACE DANS PHOTOSHOP

Les interfaces utilisateur (UI) ont été créées avec Photoshop, exportées en PNG pour une intégration dans le jeu. Des effets de flou et de profondeur de champ ont amélioré l'esthétique, tandis que la réduction du bruit a rehaussé la qualité des images. Les icônes ont été conçues pour une bonne visibilité et préparées en blanc pour faciliter les modifications dans Unity.

3.4 TECHNIQUE DE PRODUCTION AUDIO ET INTEGRATION SONORE

Dans FL Studio, les musiques ont été optimisées pour des boucles fluides avec des fondus, et l'outil Edison a été utilisé pour éliminer les bruits indésirables dans les dialogues, car un téléphone a été utilisé pour les enregistrer. Divers effets ont ensuite été ajoutés pour améliorer la qualité sonore et rendre l'écoute plus agréable.

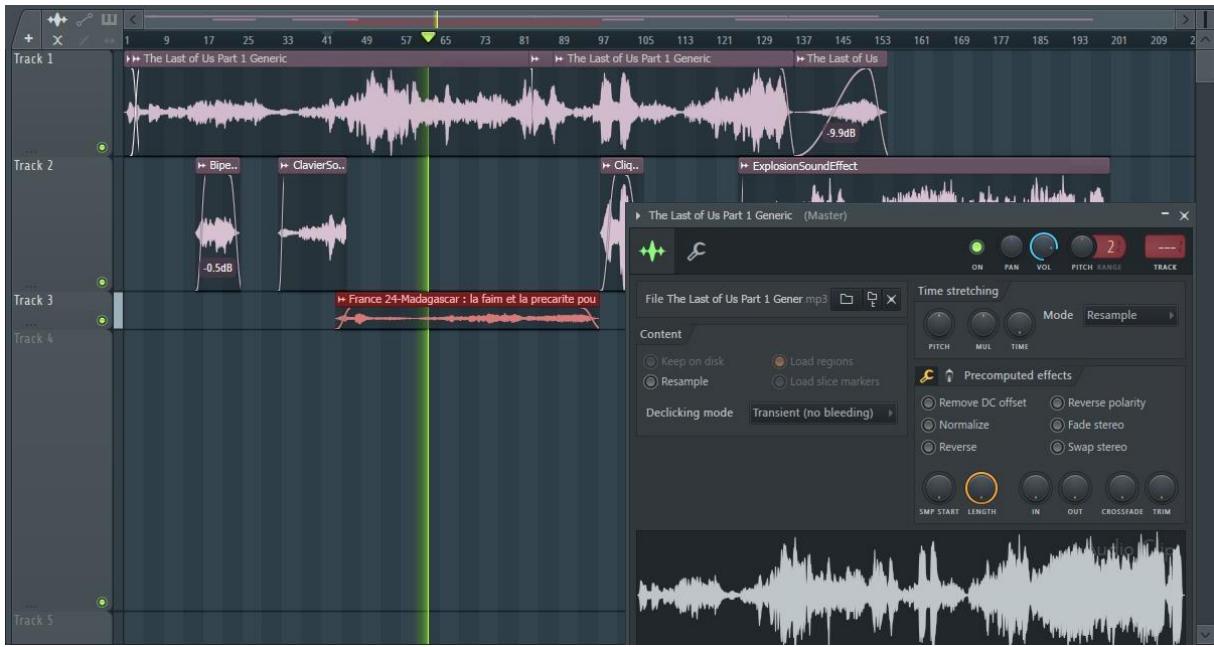


Figure (3, 22) : Arrangement audio pour la vidéo générique dans FL Studio

3.5 PRE-PRODUCTION DANS UNITY

3.5.1 Gestion du projet

Dans Unity, il existe différents types de pipelines de rendu : Built-in Render, HDRP, URP, etc. Pour ce projet, le Built-in Render a été choisi, car il est adapté aux projets de base nécessitant moins de ressources et permettant une optimisation plus simple, idéale pour un jeu ciblant une large compatibilité matérielle.

3.5.2.1 Prototypage initial

La première étape dans Unity a consisté à créer un prototype simplifié dans Blender, en utilisant des formes géométriques de base (plans, cubes, sphères). Cela a permis de valider les fonctionnalités dans Unity avant d'intégrer des modèles plus complexes. En se concentrant sur des formes simples, le projet a été géré efficacement, réduisant les temps de chargement et les problèmes de performance.

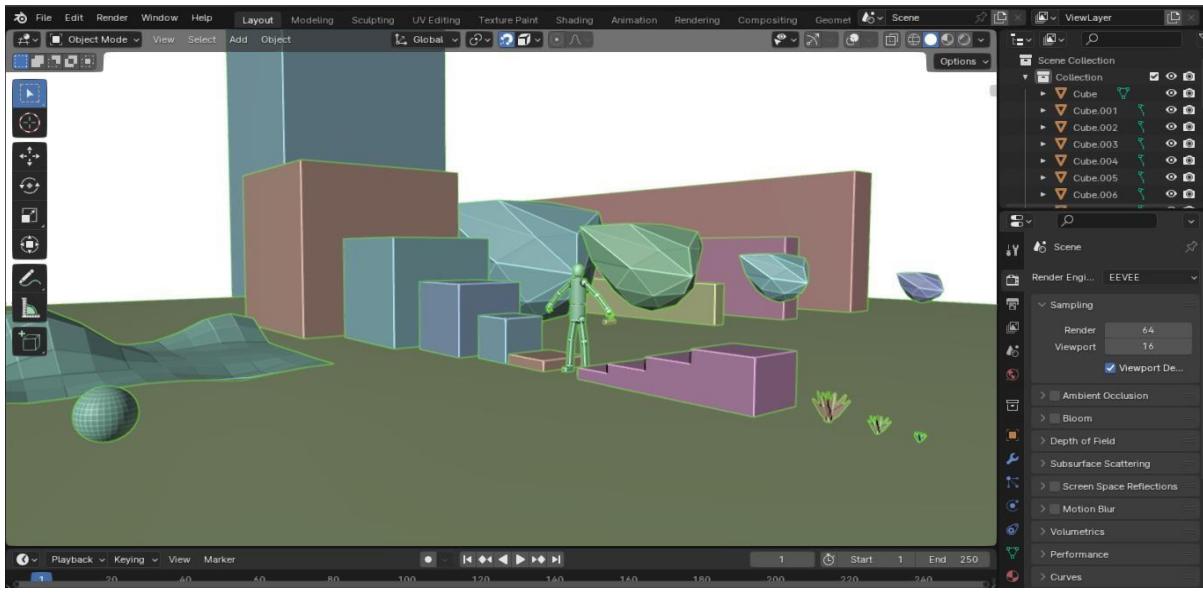


Figure (3, 23) : Formes de Base pour Prototypage

3.5.2.2 Architecture du projet

Le projet a été structuré en plusieurs scènes, chacune dédiée à une fonctionnalité, facilitant le développement et les tests. Cette approche a permis d'assembler la scène principale et d'ajuster les éléments avant l'intégration finale. Toutes les ressources (images, audio, vidéos, modèles 3D) ont été organisées dans Unity pour une gestion optimisée.

3.5.2 Package Manager

Unity permet d'accéder à divers packages via un espace de gestion. Pour le développement du jeu, sept packages clés ont été installés :

AI Navigation : Pour l'intelligence artificielle.

Animation Rigging : Pour des animations complexes.

Cinemachine : Gestion avancée des caméras.

Input System : Système d'entrée personnalisé.

Polybrush : Peinture et modélisation 3D.

Post Processing : Amélioration des effets visuels.

Text Mesh Pro : Gestion avancée du texte.

3.5.3 Outils techniques essentiels

† Layer Masks et les Tags

Les Layer Masks et les Tags sont essentiels dans Unity. Les Layer Masks regroupent les objets en couches pour optimiser les calculs de collisions, tandis que les Tags identifient des objets spécifiques, facilitant leur gestion par les scripts.

† Curves

Unity utilise des courbes pour moduler les paramètres d'un objet dans le temps, améliorant ainsi la fluidité et le réalisme des animations.

† Prefabs

Le Prefab dans Unity est un objet préconfiguré permettant de réutiliser des éléments dans plusieurs scènes, assurant cohérence et simplifiant les modifications, qui se répercutent sur toutes les instances.

3.6 POST-PRODUCTION DANS UNITY

La post-production débute par la création d'une scène 3D contenant objets, personnages et lumières, avec une caméra et une lumière directionnelle par défaut. Un Event System gère les interactions, tandis qu'un canvas organise l'interface utilisateur. Le projet comprend 8 scènes, dont certaines intègrent des vidéos via le composant Video Player, configuré pour afficher les fichiers vidéo en mode Camera View Plane avec un Aspect Ratio réglé sur Stretch pour optimiser la couverture de l'écran.

3.6.1 Réalisation de la scène START

† Script StoryManager.cs

Dans la scène "START", le script C# StoryManager.cs gère la lecture de la vidéo du logo, permettant de la passer avec un bouton et de charger la scène suivante à la fin ou après un clic. Un écran de chargement est affiché durant la transition, avec un bouton transparent, bien que le saut de cette scène ne soit pas prévu.

3.6.2 Réalisation de la scène WARNING

Dans la scène "WARNING", le script PressAnyKeyScript.cs permet de passer à la scène suivante après un délai, affichant la vidéo du capture d'écran et un fondu de l'interface

utilisateur à l'entrée. ButtonFadeIn.cs fait apparaître un bouton en contrôlant son opacité, le rendant interactif à la fin de l'animation. StoryManager2.cs fonctionne comme StoryManager.cs, sans réinitialiser la vidéo lors de son activation.

3.6.3 Composants communs des scènes BEFORE et ENDSTORY

La section relative à la réalisation présente la création des composants communs aux 2 scènes. Ces scènes incluent des éléments identiques, tels qu'un vidéo Player pour lire la vidéo générique et la vidéo de fin sur l'histoire du personnage.

Des scripts sont utilisés dans les 2 scènes y compris StoryManager.cs, ButtonFadeIn.cs et ActivateAndDeactivateGameObject.cs. Elles comportent également un bouton permettant de passer la vidéo, ainsi qu'un lecteur vidéo (Vide Player).

3.6.4 Composants et techniques communs des scènes LOBY et GAME

Dans les scènes **LOBBY** et **GAME**, des objets communs sont utilisés, incluant des caméras avec **Cinemachine**, des environnements de décors statiques et dynamiques, ainsi que des éléments d'interface utilisateur (UI) et des effets spéciaux.

3.6.5.1 Composants Cinemachine

Avec **Cinemachine**, le composant CinemachineBrain a été appliqué à la caméra par défaut, et des CinemachineVirtualCamera ont été créées. Un GameObject vide nommé "follow" est enfant du personnage pour servir de point de référence. CinemachineBrain gère les transitions de caméra, tandis que CinemachineVirtualCamera définit les paramètres de prise de vue.

3.6.5.2 Composants de l'UI d'Unity

L'interface utilisateur (UI) d'Unity comprend plusieurs composants clés pour une interaction dynamique :

Toggle : Bascule entre les états activé/désactivé avec isOn.

Button : Déclenche des actions via l'événement OnClick.

Slider : Contrôle des valeurs numériques comme le volume avec OnValueChanged.

ScrollView : Permet de naviguer dans un contenu plus grand que l'affichage.

Dropdown : Affiche une liste d'options sélectionnables.

Input Field : Capture les entrées de texte de l'utilisateur.

3.6.5.3 Importance de l' UX (User Experience)

Une bonne expérience utilisateur (UX) est essentielle dans la conception d'interfaces, car elle influence les interactions. En intégrant des éléments intuitifs, on favorise une navigation fluide, améliore la satisfaction et permet aux joueurs d'accomplir rapidement leurs tâches, garantissant ainsi une expérience de jeu agréable.

3.6.5.4 Configurations d'Animation des personnages (Player, NPC, Enemy)

⊕ Animation Type : Humanoid

Unity propose trois types d'animation : Legacy, Generic et Humanoid. Le type Humanoid est idéal pour les personnages bipèdes, permettant la réutilisation et le retargeting d'animations.

⊕ Utilisation des Finite State Machines (FSM)

Les FSM gèrent le comportement et les animations des personnages dans Unity, permettant de passer d'un état à un autre selon des conditions prédéfinies .

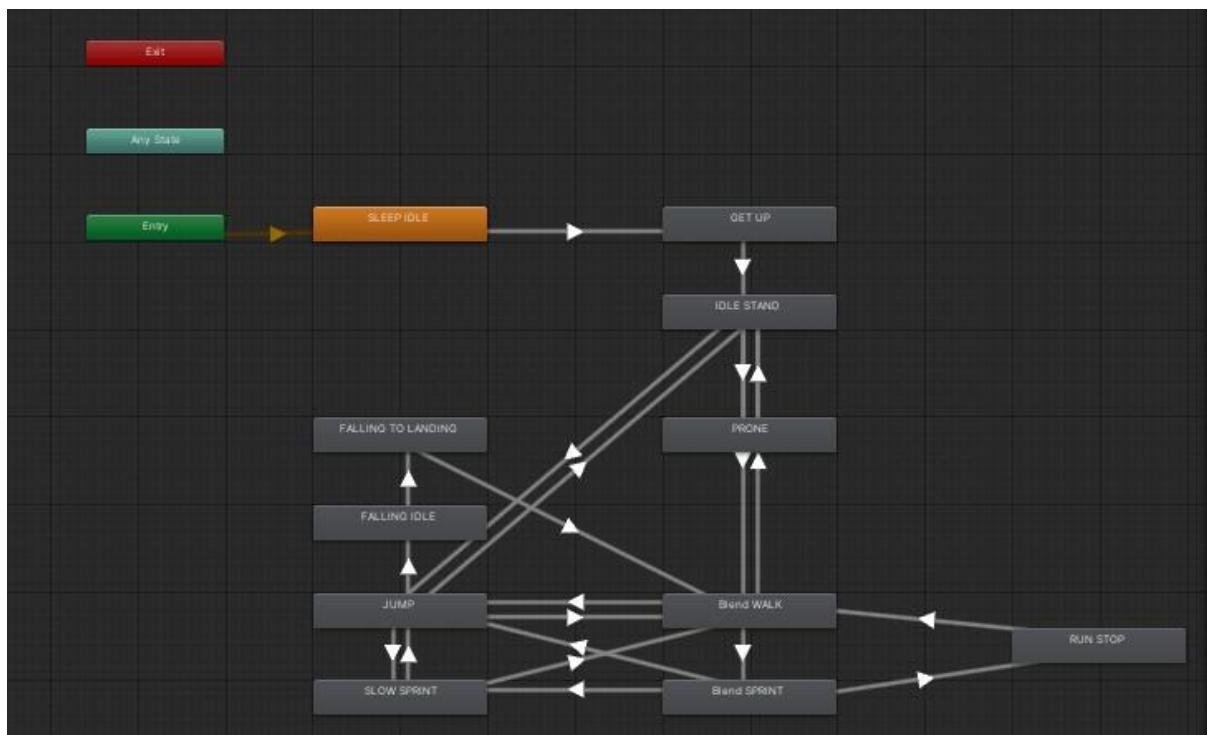


Figure (3, 24) : FSM du Player dans Animator

Parameters (Paramètres)

Les Parameters dans l'Animator contrôlent les transitions d'animation, rendant les animations réactives aux actions du joueur.

† Loop Time et Bake Into Pose

Pour les animations répétitives (comme) (voir Annexe 2), il est essentiel d'activer **Loop Time** et les options **Bake Into Pose** pour **Root Transform Rotation** et **Position (Y/Z)**. Cela maintient le modèle dans une position fixe et naturelle sans dérive.

Pour les animations non répétitives (comme le saut) (voir Annexe 3), **Bake Into Pose** est généralement utilisé uniquement pour **Position (Y/Z)**, ce qui permet de contrôler la position verticale sans fixer la rotation.

3.6.5.5 Utilisation du Système de Terrain dans Unity

Modéliser un terrain dans Blender est possible, mais le système intégré d'Unity offre des avantages en termes de gain de temps et d'optimisation. Il permet de créer facilement des paysages réalistes (montagnes, plaines, forêts) avec des outils intuitifs de sculpture et de texturisation.

† Fonctionnalités Utilisées dans le Projet :

Create Neighbor Terrain : Génère des terrains adjacents.

Paint Texture : Applique des textures pour personnaliser les surfaces.

Outils de sculpture : Inclut Set Height, Raise or Lower Terrain, Stamp Terrain, et Smooth Weight.

Paint Hole : Crée des cavités dans le terrain.

Paint Tree : Ajoute des arbres.

Paint Detail : Ajoute des éléments comme des buissons ou de l'herbe.

Terrain Settings : Configure la taille, la résolution et les optimisations.

† Configuration de la Texture avec Paint Texture

Pour texturer un terrain, un **Terrain Layer** est créé. L'utilisation des **textures PBR** (PhysicallyBased Rendering) dans la texturisation des terrains offre des avantages significatifs. Les textures PBR permettent de simuler des matériaux de manière plus réaliste en ajustant des paramètres tels que la **Base Color**, le **Normal Map**. Le **Normal Map** crée une illusion de relief sur des surfaces planes, en modifiant la direction des normales des pixels pour simuler des détails comme des bosses ou des creux, ce qui donne l'impression d'une surface plus complexe sans augmenter la géométrie. Cette technique améliore également l'interaction avec

la lumière, rendant les terrains plus dynamiques, et permet de simuler des effets de variation de surface.



Figure (3, 25) : Visualisation des textures PBR

† Création d'Arbres dans Unity

Dans Unity, le composant Tree intégré permet de créer et de personnaliser des arbres sans modélisation manuelle. Il propose des options comme Add Branch Group et Add Leaf Group pour un contrôle total sur leur apparence.

Problème de Collision

Les arbres sur le terrain n'ont pas de colliders par défaut, permettant aux objets de les traverser. Pour corriger cela, des GameObjects vides avec des Capsule Colliders ont été ajoutés autour des arbres, et l'ensemble a été regroupé dans un Prefab pour faciliter la réutilisation.

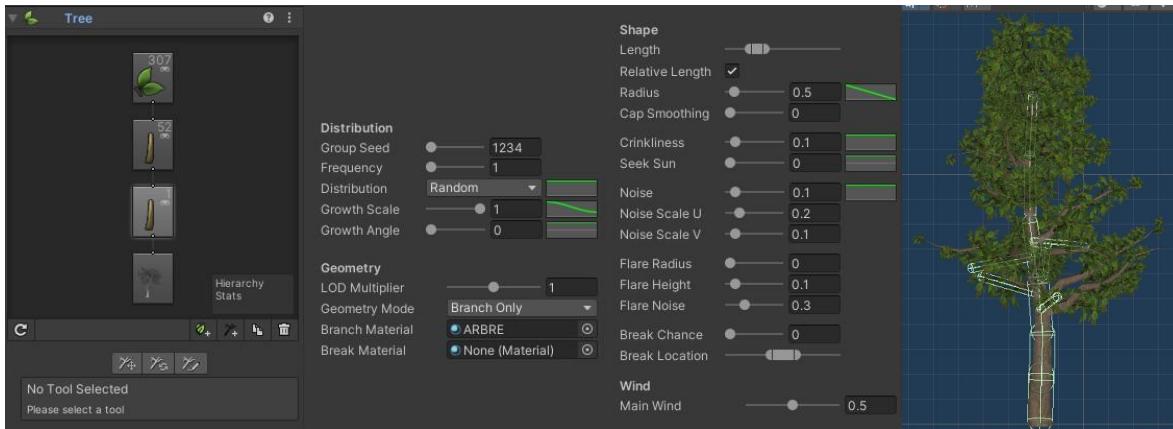


Figure (3, 26) : Composant Tree et ses colliders

⊕ Création d'herbe dans Unity

Pour l'herbe, un Mesh a été utilisé avec le composant Tree d'Unity sans branches.

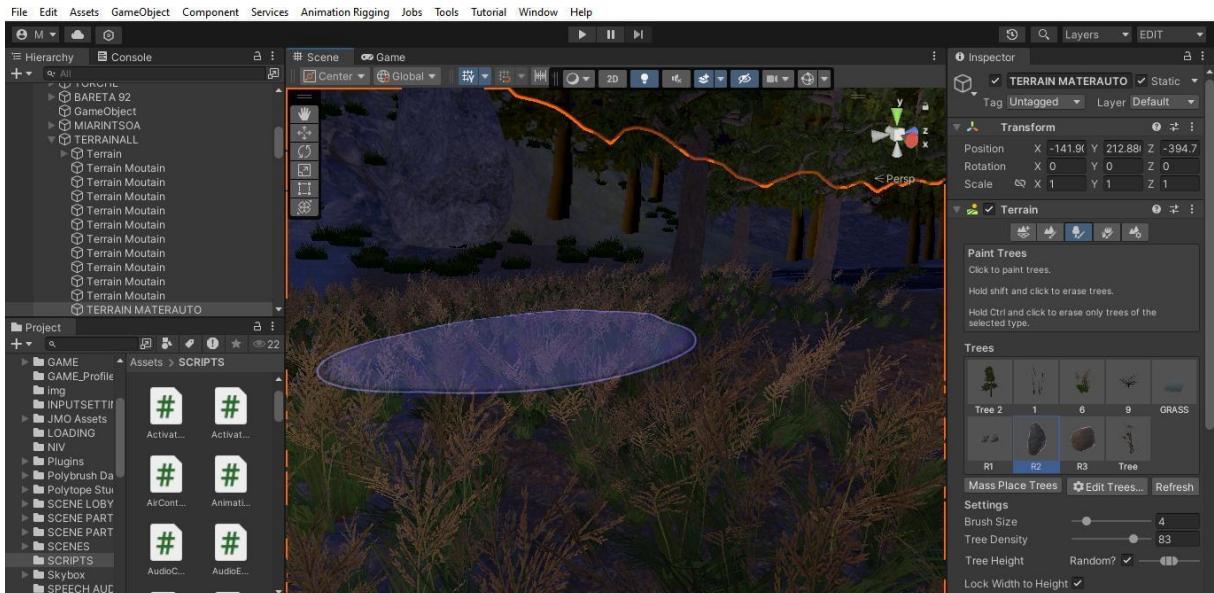


Figure (3, 27) : Composant Terrain

3.6.5.6 Audio Source et Audio Listener

L'audio dans Unity utilise deux composants principaux : l'Audio Source pour émettre des sons et l'Audio Listener pour les capter, généralement attaché à la caméra. Le **Spatial Blend** ajuste la transition entre audio 2D et 3D, améliorant l'immersion. Les réglages incluent :

Radius : Zone d'audibilité.

Min Distance : Distance pour le volume maximal.

Max Distance : Portée d'atténuation.

Doppler : Influence de la vitesse sur la fréquence perçue.

3.6.5 Réalisation de la scène LOBY

La scène lobby contient les boutons Jouer, Chapitre, Info, Quitter et Crédits. Le bouton Quitter exécute un script QuitGame, utilisant la fonction Application.Quit pour fermer l'application. Le bouton Jouer ouvre une interface (UI) comprenant un bouton Continuer, relié au script LobbyManager , qui reprend les fonctions du StoryManager tout en réduisant progressivement le volume de la musique de fond via une liste de réglages. Le bouton Info affiche les instructions du jeu, le bouton Chapitre montre les chapitres disponibles, et le bouton Crédits lance une animation de crédits.

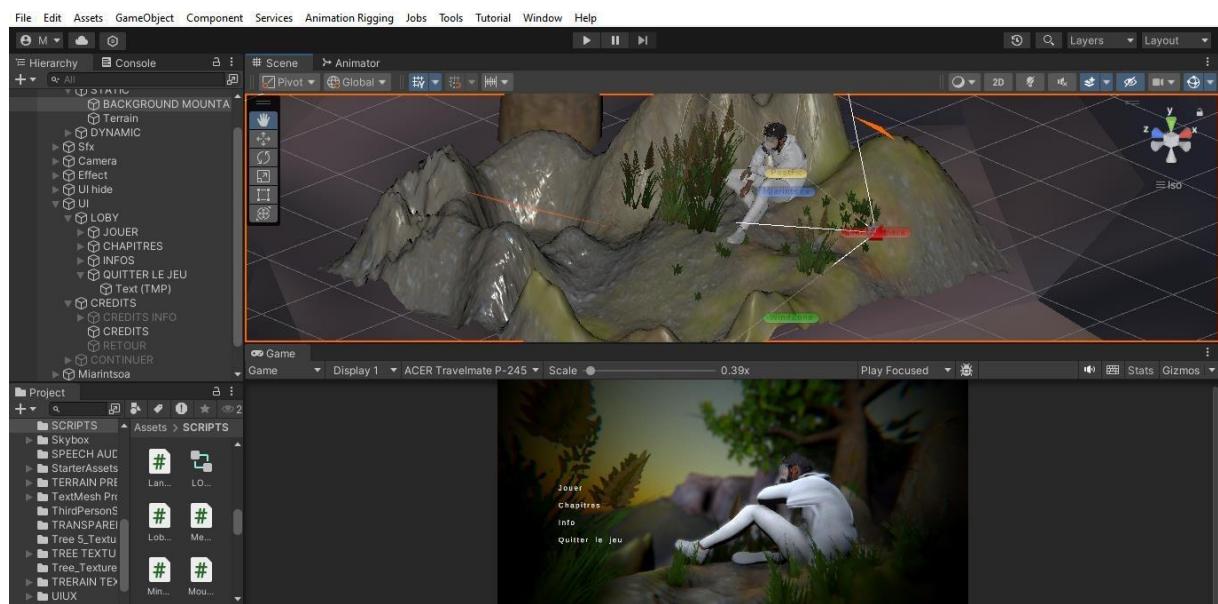


Figure (3, 28) : Aperçu de la scène Loby

3.6.6 Réalisation de la scène GAME

3.6.6.1 Le personnage controlable

Le contrôle du personnage jouable repose sur une intégration plusieurs composants et plusieurs scripts pour garantir une jouabilité fluide.

† Le composant CharacterController

CharacterController gère les déplacements et les collisions sans nécessiter de Rigidbody. Le paramètre Slope Limit détermine l'angle maximal de pente que le personnage peut franchir, ce qui est essentiel pour les terrains accidentés. Un réglage optimal du Slope Limit, situé entre 30° et 45°, équilibre réalisme et accessibilité en empêchant le personnage de gravir des pentes trop raides.

† PlayerInput (Package InputSysteme)

Le PlayerInput facilite la gestion des entrées du joueur pour les actions telles que le déplacement, le saut, et d'autres interactions.

† Animator

L'Animator dans Unity gère les animations du personnage (marche, course, saut) en utilisant un avatar pour définir la structure du squelette.

† RigBuilder et BoneRenderer (Package AnimationRigging)

RigBuilder et BoneRenderer sont utilisés pour gérer les animations basées sur l'inverse kinematics (IK). RigBuilder configure les rigs d'animation pour ajuster automatiquement les articulations du personnage en fonction des mouvements, tandis que BoneRenderer affiche les os dans l'éditeur, facilitant l'ajustement visuel des systèmes d'IK.

† AudioSource

Gère la lecture des effets sonores et des ambiances.

† Scripts C# attacher au personnage Player IntroScript.cs

Le script gère l'introduction où le personnage commence "endormi". Tous les scripts de mouvement sont désactivés, et l'animation "he's sleeping" est activée. Après un délai (standUpDelay), une invite "Appuyez sur E" apparaît. Lorsque le joueur appuie sur "E", le personnage se lève (StandUp()), et l'animation "he's standing up" est jouée. Une fois l'animation terminée, les scripts de mouvement sont réactivés et l'introduction se termine.

StarterAssetsInputs.cs

Le script gère les entrées utilisateur via le système d'entrée d'Unity pour les mouvements et actions (sprint, visée, saut, accroupissement,etc). Il capture et stocke les valeurs d'entrée, simplifiant le contrôle du personnage. Il gère aussi le verrouillage du curseur selon le focus du jeu.

ThirdPersonController.cs

Le script gère les mouvements du personnage, permettant de marcher, courir, s'accroupir et sauter, tout en synchronisant les animations. Il limite l'angle de rotation de la caméra et réduit la vitesse après 5 secondes de sprint, jouant l'animation "MediumSprint".

CrouchController.cs

Le script ajuste la hauteur du CharacterController en fonction de l'état "Crouch" de l'Animator. Lorsque le personnage est accroupi, la hauteur est réduite, sinon elle revient à la hauteur originale.

PlayerAudioBreath.cs

Le script PlayerAudioBreath.cs adapte les sons de respiration à la vitesse du personnage, jouant idleBreath en dessous de 1 unité et fastSprintBreath au-dessus de 5 unités, tout en évitant la superposition des clips audio.

AudioController.cs

Le script AudioController gère les sons de pas et d'atterrissement. Il contient une liste de clips audio pour les pas (FootstepAudioClips) et un clip pour l'atterrissement (LandingAudioClip), joués à un volume défini par FootstepAudioVolume.

- **PlayFootstep** sélectionne et joue un son aléatoire de pas.
- **PlayLandingSound** joue le son d'atterrissement.

Des événements sont ajoutés dans les animations via l'Animator : ils déclenchent PlayFootstep à chaque pas dans l'animation de marche et PlayLandingSound à l'atterrissement, synchronisant ainsi les sons avec les mouvements du personnage.

AnimationController.cs

Le script gère les animations d'un personnage via l'Animator en mettant à jour les paramètres selon l'état du personnage (vitesse, saut), récupérant le composant Animator et assignant des identifiants via StringToHash, avec des méthodes clés comme SetSpeed et SetJump.

ThirdPersonViewController

Ce script gère le basculement entre une vue à la troisième personne et une vue à la première personne. Lorsqu'une entrée utilisateur (toggleView) est détectée, il active ou désactive la caméra appropriée. Il calcule également la position du curseur dans le monde, désactivant les éléments de débogage en vue à la première personne.

La Caméra

Avec le package **Cinemachine** dans Unity, le composant **CinemachineBrain** est attaché à la caméra principale pour gérer les transitions entre différents états de caméra, comme **PlayerCameraThirdPerson**, **PlayerCameraAim**, et **FirstPersonCamera**.

Un GameObject "**follow**", enfant du personnage, sert de référence pour la caméra, permettant ainsi de suivre le joueur de manière fluide. Les **effets de noise** dans Cinemachine sont utilisés pour ajouter des vibrations ou des secousses à la caméra, créant une sensation de réalisme.

3.6.6.2 Systèmes de vies (PlayerHealth.cs)

Le script PlayerHealth gère la santé du joueur, en appliquant des dégâts provenant de différentes sources, comme les zones de feu, les chutes, les balles et les coups de poingt. Lorsqu'un joueur subit des dégâts, sa santé est mise à jour et affichée via une barre de santé (Slider). Si la santé atteint zéro, le joueur meurt tout en désactivant tous les scripts nécessaires, ainsi que l'Animator. Le script affiche également un panneau "Game Over" et déverrouille le curseur de la souris. En outre, il applique des dégâts de manière continue dans les zones de feu, gère les dégâts de chute en fonction du temps passé dans les airs, et met à jour l'UI en fonction de la santé du joueur.

3.6.6.3 AI (Intelligence Artificiel)

† **Création des Animations pour les Ennemis et les NPCs dans Mixamo** Pour les animations des ennemis et NPCs, Mixamo a été utilisé avec Blender pour gagner du temps. Les modèles ont d'abord été placés en semi T-pose dans Blender pour assurer une bonne détection des os par Mixamo. Sur Mixamo, les personnages importés en FBX sont paramétrés pour les articulations principales, puis les animations de déplacement sont exportées en mode In Place pour maintenir le modèle en position fixe.

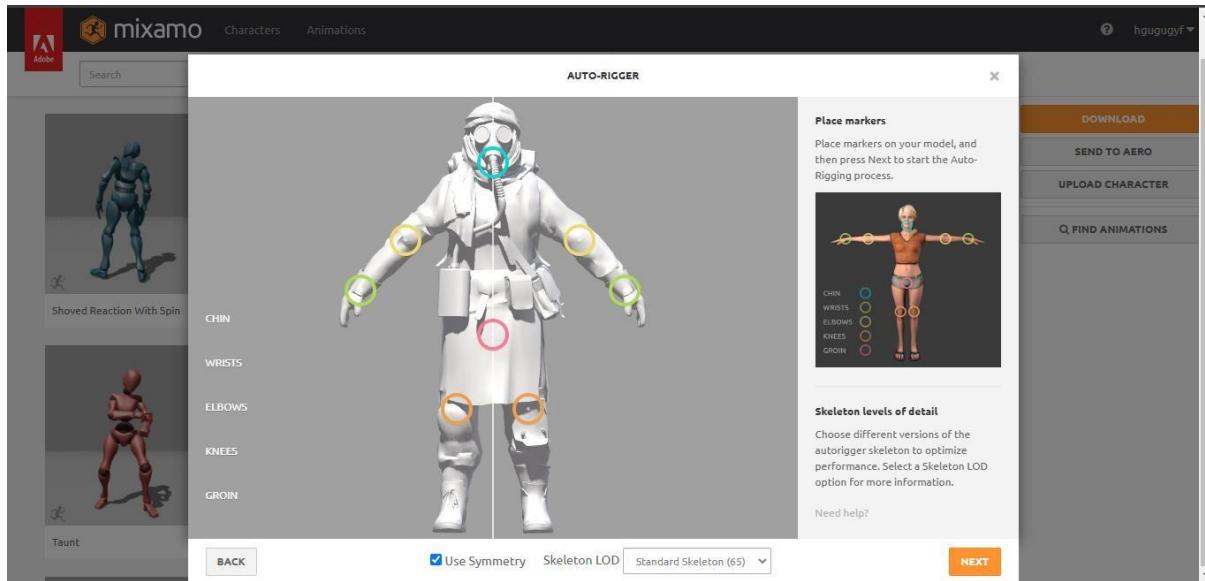


Figure (3, 29) : Placement des markers sur le model NPC dans Mixamo

† Utilisation du NavMesh (NavigationMesh)

Le **NavMesh** d'Unity est une surface virtuelle utilisée pour le **pathfinding**, permettant aux personnages de se déplacer en suivant le chemin optimal et en évitant les obstacles. En générant un NavMesh sur la scène, les agents peuvent calculer leur trajet en utilisant le composant **NavMesh Agent**.

Dans ce projet, un **seul NavMesh Surface** est utilisé par toutes les intelligences artificielles (IA), ce qui permet d'optimiser les performances en évitant de créer plusieurs surfaces pour chaque IA. Chaque ennemi utilise un **NavMesh Agent** pour se déplacer de manière autonome, et le système de pathfinding assure des déplacements réalistes en fonction de la géométrie de la scène.

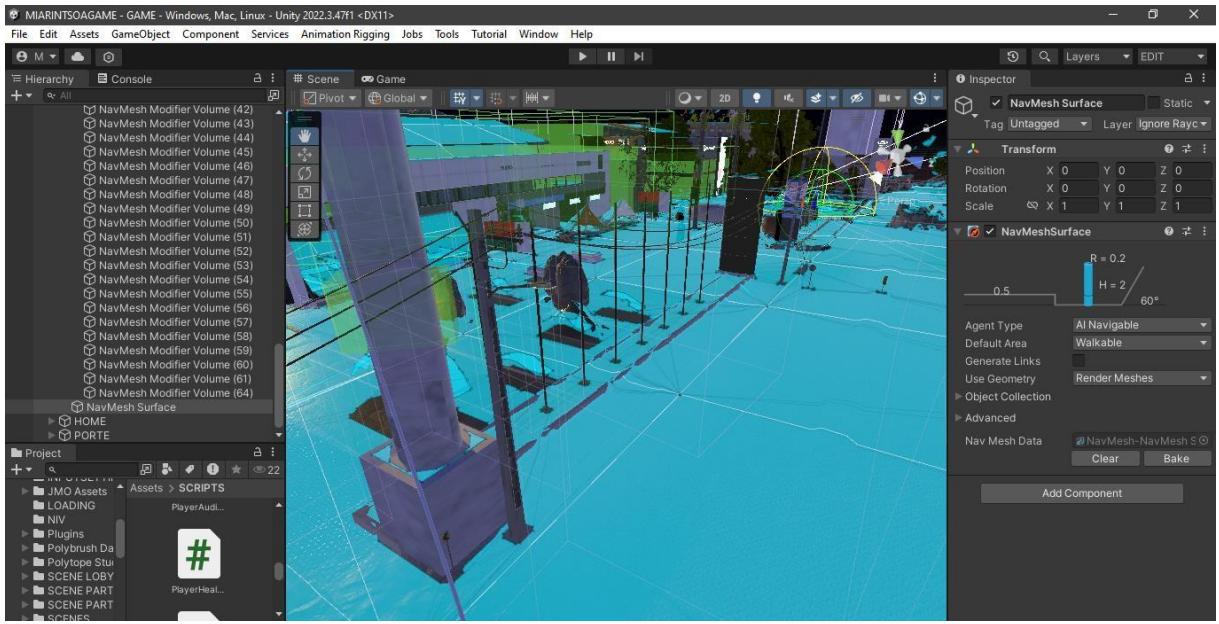


Figure (3, 30) : Surface navigable des IA

† Fonctionnement du NPC

Dans le jeu, le NPC a pour rôle de guider le joueur vers un point précis. Un script **NPCController** gère le comportement du NPC (personnage non-joueur) en fonction de sa proximité avec le joueur. Utilisant le **NavMeshAgent** pour les déplacements, le script ajuste la vitesse de l'animation et les actions du NPC selon trois rayons de détection définis. Le NPC réagit selon la distance du joueur : à proximité (closeRadius), il s'arrête et cesse de bouger ; à distance moyenne (mediumRadius), il se dirige vers un point cible avec une animation de marche ; au-delà du rayon de détection (detectionRadius), il se contente d'observer sans se déplacer.

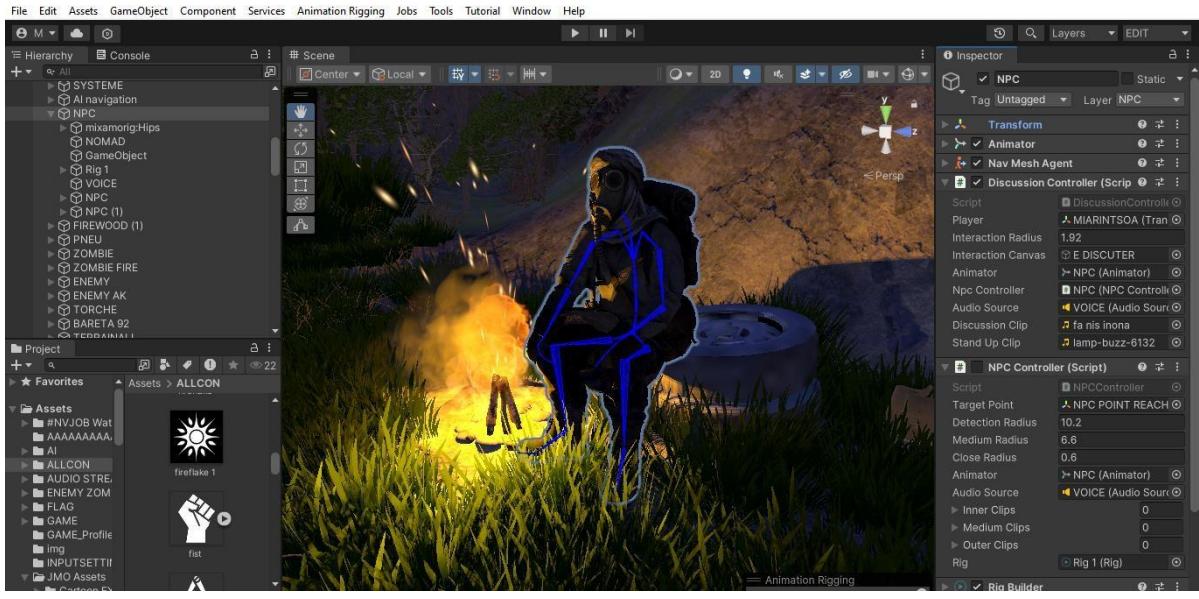


Figure (3, 31) : Personnage non joueur NPC

† Fonctionnement des ennemis

Le script **EnemyInteractionWithPlayer.cs** gère l'interaction des ennemis avec le joueur en utilisant un **NavMeshAgent** pour les déplacements et un **Animator** pour les animations basées sur la distance. L'ennemi patrouille, détecte le joueur à différentes distances (marche, course, attaque), et ajuste sa vitesse et son animation en conséquence. Lorsqu'il entre dans la zone d'attaque, il se tourne vers le joueur, le poursuit et joue l'animation d'attaque. La gestion des layers dans l'**Animator** assure une transition fluide entre les animations, tandis que les **Gizmos** visualisent les zones de détection dans l'éditeur. Le script **EnemyAKInteractionWithPlayer.cs** permet à l'ennemi de suivre le joueur et de tirer lorsqu'il est à portée. Il applique des effets visuels et sonores lors de chaque tir. Lors de l'entrée du joueur dans la zone de tir, l'ennemi s'oriente vers lui, tire à intervalles réguliers, et ajuste le poids du layer d'animation en fonction de l'état de tir. Les balles sont projetées avec un **Rigidbody** pour appliquer la direction et la vitesse.

Ennemi avec bâton :

Un ennemi équipé d'un bâton (enfant du rig `mixamo_RightHand`) utilise le script `PlayerHealth` pour infliger des dégâts au joueur. Le bâton, ayant le tag `Weapon`, réduit la vie du joueur de 10 points à chaque collision. Cependant, les dégâts augmentent progressivement avec le nombre de coups reçus, simulant un effet cumulatif dû aux multiples frappes.

Ennemi avec AK-47 :

Les ennemis armés d'un AK-47 utilisent des **balles** qui suivent un comportement similaire à celui de l'arme Barreta du joueur. Un seul coup de feu suffit à tuer le joueur. Les balles ennemis ont également un impact sur l'arme du joueur selon son état, utilisant un système de gestion similaire à celui des armes du joueur.

Ennemi zombie et zombie en feu :

Il existe deux types de zombies : les zombies normaux et les zombies en feu. Bien qu'ils partagent les mêmes scripts et fonctionnalités, leurs zones de détection diffèrent. Les zombies en feu détectent les joueurs à une plus grande distance et infligent davantage de dégâts à la vie du joueur que les zombies normaux.



Figure (3, 32) : Aperçu des différents types d'ennemis

Simulation de balles réelles :

Pour simuler le comportement réaliste d'une balle, les armes (utilisées par le joueur ou les ennemis) ne visent pas directement le point désigné, introduisant une légère dispersion pour plus de réalisme. Le prefab de la balle intègre un **Rigidbody** avec l'option **Interpolate** activée pour un mouvement fluide, et la détection de collision réglée sur **Continuous Dynamic**. Cette configuration garantit un déplacement rapide et précis de la balle dans l'espace tout en évitant les erreurs de collision ou le phénomène de "tunneling".

BulletProjectiles.cs

Le script **BulletProjectile** permet à la balle de se déplacer à une vitesse définie, de détecter les collisions avec des objets cibles ou autres, et d'instancier des effets visuels spécifiques selon le type de collision. Si la balle touche un objet avec le composant **BulletTarget**, un effet de sang est créé, sinon un effet d'impact est généré. Enfin, la balle est détruite après la collision.

BulletTarget.cs

Le script **BulletTarget** est attaché à tous les personnages du jeu, il gère l'interaction d'un personnage avec une balle. Lorsqu'une balle entre en collision avec l'objet (vérifié par OnTriggerEnter), il désactive plusieurs composants du personnage pour simuler des dégâts. Il désactive l'Animator pour stopper les animations, le Collider pour éviter des collisions futures, et le NavMeshAgent pour empêcher l'ennemi de se déplacer. De plus, tous les scripts ajoutés à la liste scriptsToDisable sont également désactivés, ce qui peut inclure des comportements de l'ennemi ou d'autres systèmes. Cela permet de simuler une mort ou une incapacité du personnage à agir après avoir été touché par une balle.

Détection des balles rapides et ajout du second collider

Dans Unity, la détection des collisions peut poser problème pour les objets rapides, comme les balles, qui peuvent passer à travers d'autres objets. Pour résoudre cela, un **deuxième collider** a été ajouté au **bullet prefab**. Ce collider permet de forcer la détection des collisions, assurant ainsi que les balles interagiront correctement avec les surfaces et le joueur, même à grande vitesse.

3.6.6.4 Éléments d'interaction

Equipement ramassable ↗ L'arme à feu Beretta92

Après avoir modélisé l'arme Beretta 92 dans Blender, plusieurs éléments ont été intégrés pour la rendre fonctionnelle dans Unity. Un Muzzle Flash a été ajouté pour simuler le flash au bout du canon lors du tir. Une copie modifiée de l'arme, avec un effet d'émission lumineuse, indique visuellement l'emplacement de l'arme. Un point de sortie, appelé Spawn Bullet, a été créé pour les balles, avec une balle prefab prête à être instanciée lors du tir. Enfin, un AudioSource a été intégré pour jouer les sons du tir et du chargeur vide.

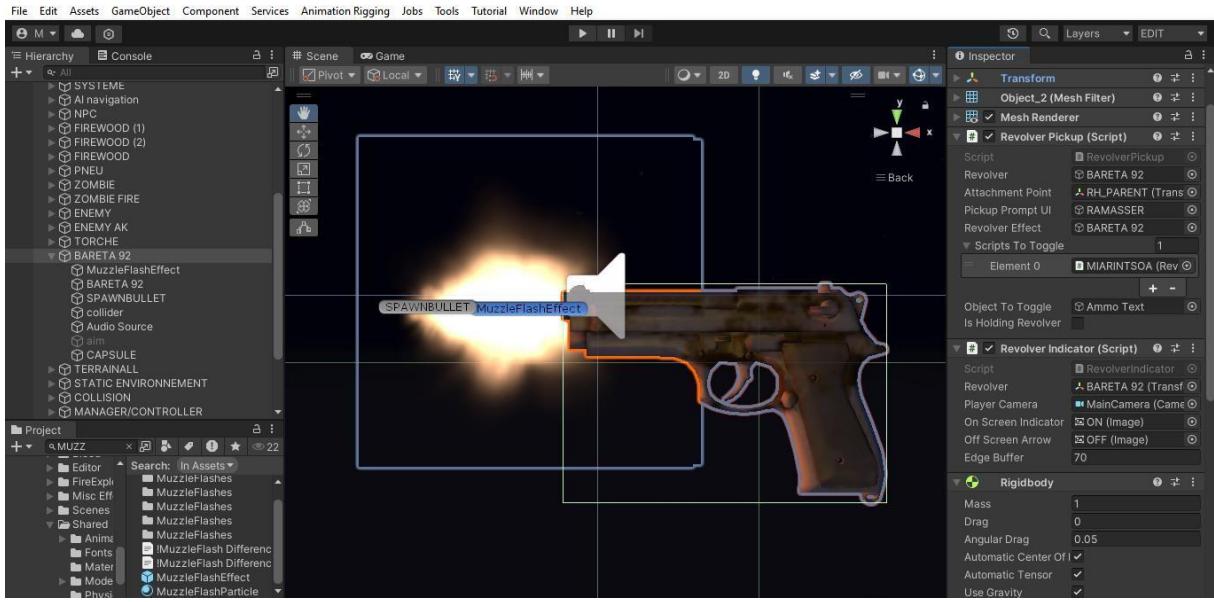


Figure (3, 33) : Equipement arme à feu (Baretta 92)

BarettaPickup.cs

Le script gère le ramassage et le lancement de l'arme dans le jeu : lorsque le joueur est proche de l'arme, une invite à l'écran lui demande d'appuyer sur "F" pour la ramasser. Une fois ramassée, l'arme est attachée à un point spécifique sur le joueur, et certains scripts et objets sont activés ou désactivés en fonction de l'état de l'arme. Lorsque le joueur appuie à nouveau sur "F", l'arme est lâchée et renvoie un effet visuel tout en réactivant les scripts associés au joueur.

BarettaIndicator.cs

Le script permet d'afficher un indicateur à l'écran pour l'arme en fonction de sa position par rapport à la caméra du joueur. Si l'arme est visible, un indicateur à l'écran suit sa position, et si elle est hors de l'écran, une flèche directionnelle apparaît pour indiquer sa position relative. Ce mécanisme prend en compte l'orientation de l'arme et ajuste la position de la flèche en conséquence. Si le joueur tient l'arme, les indicateurs UI sont désactivés.

BarettaAimController.cs

Le script permet de gérer la visée et le tir d'un revolver dans un jeu : il active la caméra de visée et ajuste la sensibilité lors de la visée, gère le tir en vérifiant les munitions, déclenche des effets visuels et sonores, met à jour l'affichage des munitions, désactive certains colliders pendant la visée, et affiche un message quand le chargeur est vide.

† Lamp torche (Flashlight)

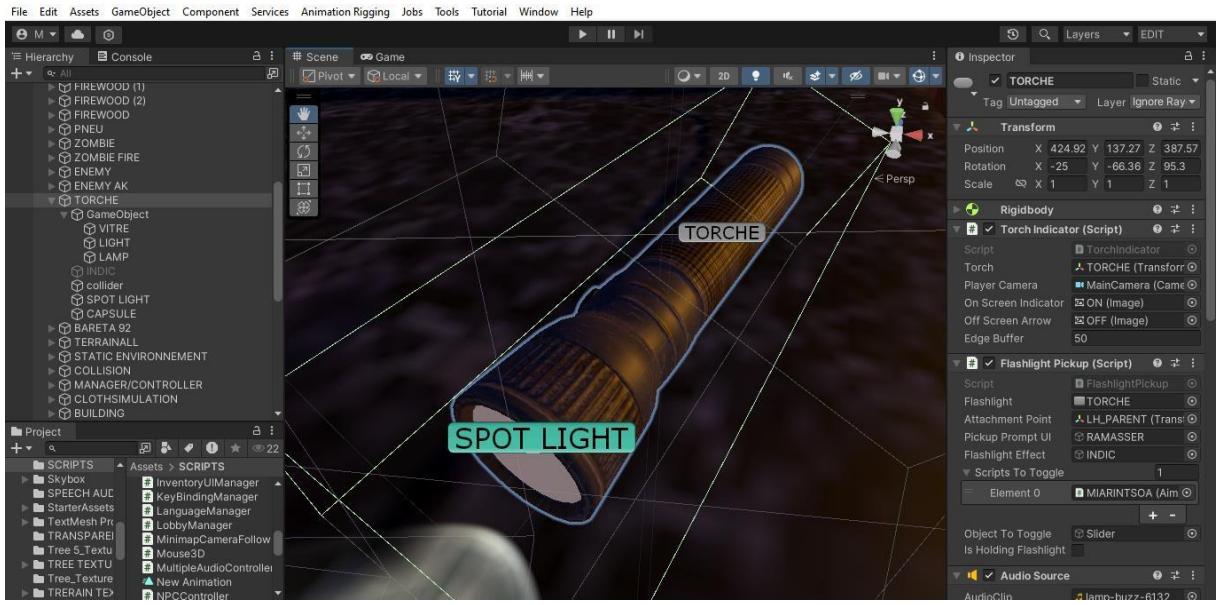


Figure (3,34) : Equipement lampe torche

FlashlightAimController.cs

La lampe torche fonctionne de manière similaire à l'arme Beretta. Les mêmes scripts utilisés pour la Beretta ont été appliqués, à l'exception du script **BerettaAimController**. Ici, le script **FlashlightAimController** fonctionne de manière analogue au **BerettaAimController**, mais se concentre sur l'activation de la lumière **Spot Light** de la lampe torche. Plutôt que l'action "Shoot", il utilise l'action "Aim" pour allumer la lampe torche. De plus, un slider a été intégré pour afficher la durée de vie restante de la batterie. Lorsque cette durée atteint zéro, la lampe torche s'éteint automatiquement, simulant ainsi la gestion de la batterie.

3.6.6.5 Paramètres du jeu

Un paramètre de jeu a été créé pour gérer certaines interactions dynamiques au sein de l'interface utilisateur. Dans Unity, pour des éléments de l'UI tels que les boutons, sliders, toggles, etc., les méthodes des scripts peuvent être attachées aux événements, comme **onClick**, via l'inspecteur ou être contrôlées directement dans le script. En affectant dynamiquement l'UI et les événements à utiliser, le script peut gérer les interactions sans nécessiter d'assignation manuelle dans l'inspecteur.

Paramètres gameplay

Dans la section des paramètres gameplay du jeu, le joueur peut ajuster divers éléments pour personnaliser son expérience, tels que l'intensité de l'éclairage avec le Lightning Multiplier via le script

EnvironmentLightingControl.cs, activer ou désactiver la Rotation Map et d'afficher ou cacher la carte avec Hide Map via le script **MinimapCameraFollow.cs**, régler la sensibilité générale ainsi que celle utilisée pour viser avec la lampe et le revolver, activer ou désactiver le réticule avec Crosshair, et contrôler l'affichage de l'interface utilisateur avec UI Visibility.

Paramètres input

Le script **KeybindManager.cs** permet aux joueurs de personnaliser les commandes du jeu directement via l'interface utilisateur (UI). Il gère les réattributions de touches pour des actions comme le saut, le sprint, ou l'ouverture de l'inventaire. Lorsqu'un joueur clique sur un bouton, le script attend une nouvelle touche, vérifie si elle est déjà utilisée et met à jour l'affichage en conséquence. De plus, un écran avec une image et un texte informe l'utilisateur pendant le processus de reconfiguration. Il offre aussi la possibilité de réinitialiser les touches aux valeurs par défaut. Cette approche permet une gestion flexible et interactive des contrôles, améliorant l'accessibilité du jeu.

Paramètres audios

Les paramètres audios ont été implémentés pour contrôler le volume du son dans le jeu. Des scripts ont été créés afin d'ajuster ce volume à l'aide d'un slider, permettant de modifier les niveaux sonores des audios en 3D et en 2D. Ces ajustements concernent principalement le volume, sans modification des autres caractéristiques des audios.

3.6.7 Renforcement du réalisme du gameplay

3.6.7.1 Réalismes Visuel

⊕ Post-Processing Layer et Post Processing Volume

Le Post-Processing Layer, ajouté à la caméra principale, applique des effets visuels comme l'anti-aliasing et la correction des couleurs aux éléments d'un layer spécifique. Le Post Processing Volume définit les zones où ces effets sont actifs, avec des profils ajustables pour divers effets (color grading, motion blur, ambient occlusion, etc). Configuré globalement ou localement, il permet des transitions visuelles immersives, renforçant l'atmosphère du jeu.

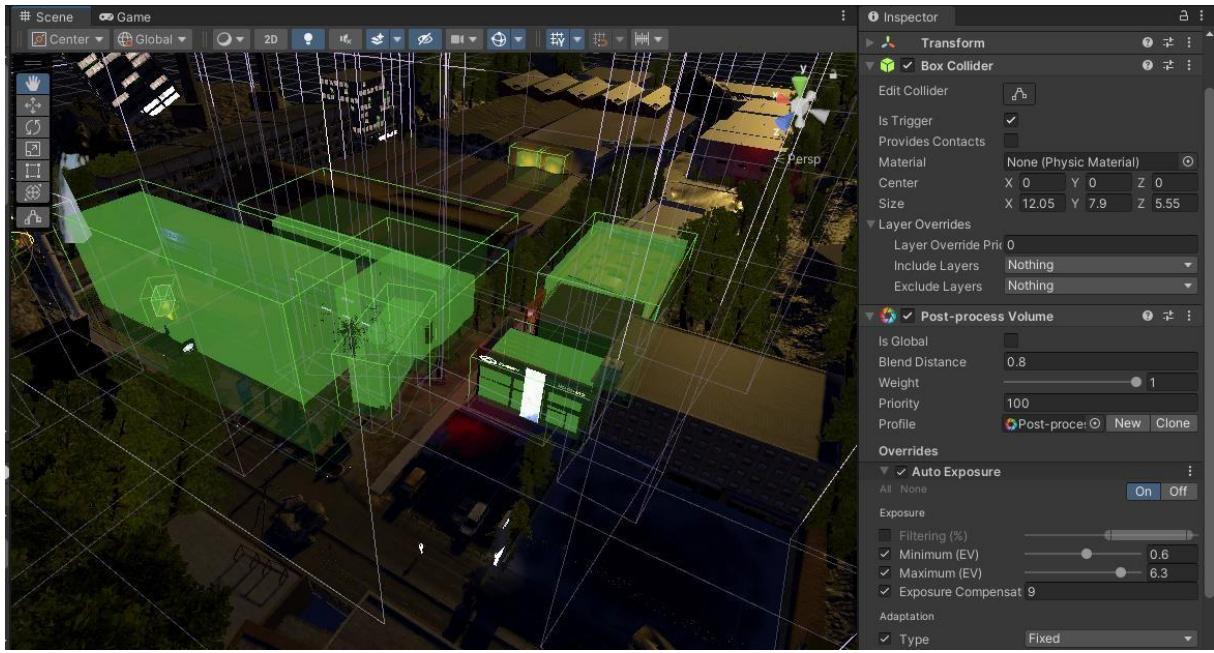


Figure (3, 35) : Composant Post Processing Volume

⊕ Utilisation des systèmes de particules

Le système de particules d'Unity permet de générer des effets visuels dynamiques à l'aide d'images planes, simulées en 3D. Il offre une large gamme de paramètres, tels que le bruit (noise), la randomisation et la gestion des forces, permettant de créer des effets variés et réalistes.

Les effets possibles incluent des chutes d'eau, de la pluie, des flammes, de la fumée, des étincelles, des éclairs, ainsi que des éclats lumineux générés par les tirs d'armes. Chaque effet peut être ajusté en modifiant des paramètres comme la taille, la vitesse, la direction, la couleur et la durée des particules, offrant ainsi une grande flexibilité pour simuler des événements naturels ou des actions en jeu.

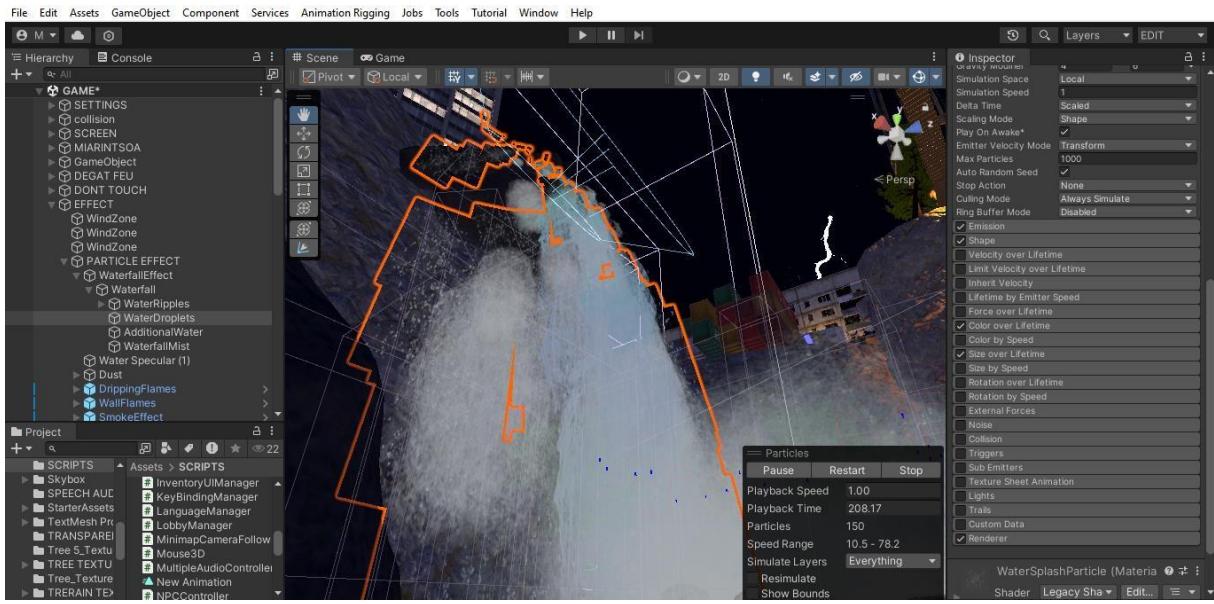


Figure (3, 36) : Aperçu du particule système pour l'effet chute d'eau

⊕ Fog (Brouillard)

Le Fog dans Unity intensifie l'atmosphère de la scène post-apocalyptique en limitant la visibilité et en créant une ambiance immersive. Il masque également les effets d'optimisation comme les billboards tree et le Far Clip Plane, atténuant les transitions en arrière-plan pour une scène visuellement cohérente et performante.

⊕ WindZone

Le WindZone dans Unity simule les effets du vent sur des éléments naturels comme les arbres et l'herbe, renforçant le réalisme visuel. Deux types existent : le Directional pour un vent constant et le Spherical pour un vent localisé qui diminue avec la distance. Pour ce projet, le Spherical WindZone a été choisi pour ajuster l'intensité et la direction du vent dans des zones spécifiques, créant des variations atmosphériques immersives adaptées aux caractéristiques de chaque emplacement.

⊕ Utilisation du composant Lens Flare

Le composant Lens Flare dans Unity ajoute du réalisme en simulant les éclats lumineux des sources intenses comme le soleil et les éclairages de pylônes électriques. Cet effet visuel renforce l'immersion en accentuant la présence des lumières dans le champ de vision.

3.6.7.2 Réalismes Technique

† Utilisation du Blend Tree

Les **Blend Trees** dans Unity permettent de combiner plusieurs animations en fonction de paramètres tels que la vitesse ou la direction du personnage. Cela crée des transitions fluides entre différents états de mouvement, comme la marche, la course ou le sprint. En ajustant ces paramètres, le personnage passe naturellement d'une animation à une autre, offrant une expérience de jeu plus réaliste et immersive.

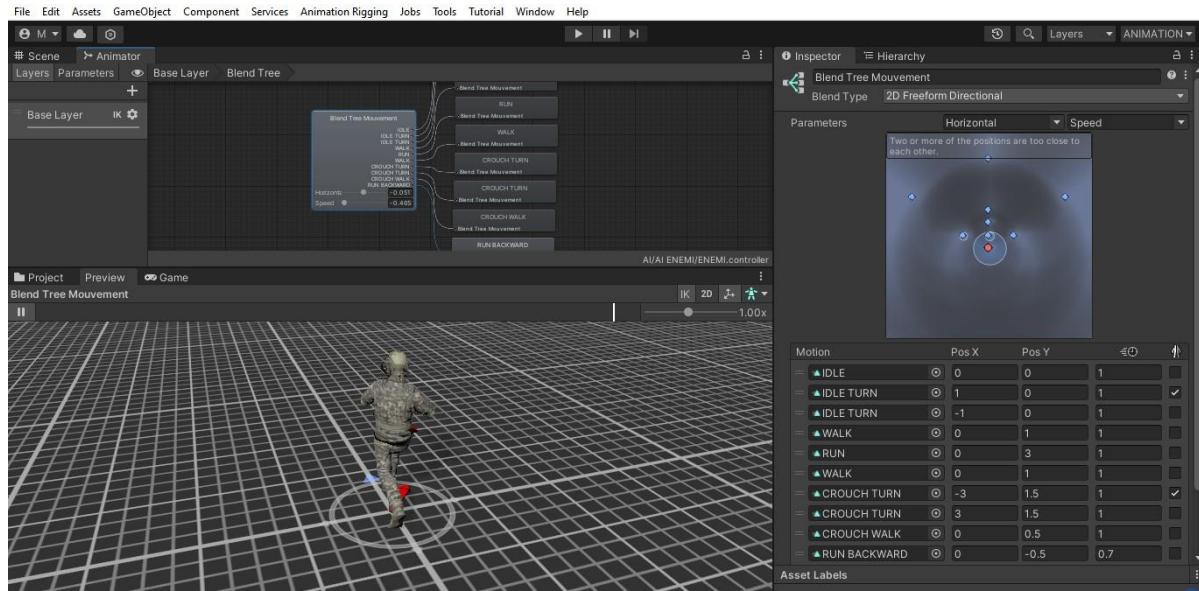


Figure (3, 37) : Blend Tree du mouvement de l'enemi avec l'arme AK47

† Layers et Avatar Masks

Les **layers** dans Unity permettent de gérer plusieurs animations sur un personnage en fonction des parties du corps. Par exemple, le **Base Layer** gère les mouvements principaux (comme courir ou sauter), tandis que d'autres layers, comme gèrent des animations spécifiques.

Les **avatar masks** définissent quelles parties du corps sont affectées par chaque layer. Par exemple, dans **RevolverAim**, l'avatar mask peut être configuré pour animer seulement le haut du corps (visée), tandis que le bas du corps reste libre pour les déplacements.

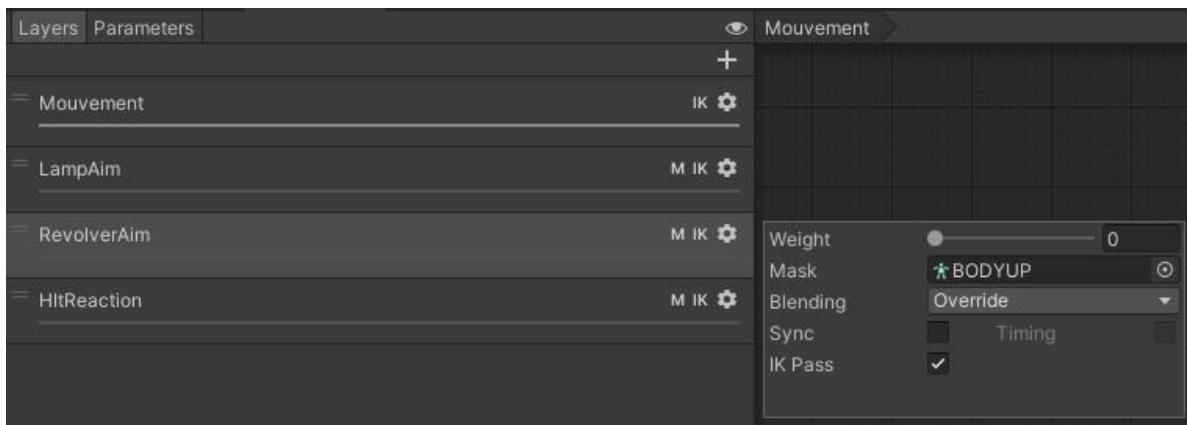


Figure (3, 38) Aperçu des Layer du Player dans Animator

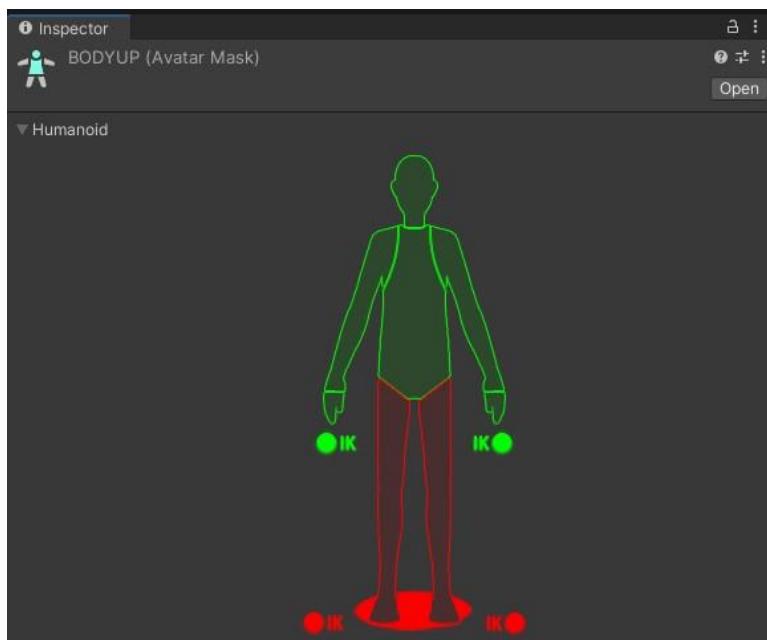


Figure (3, 39) : Avatar Mask du haut du corps

En modifiant le **poids (weight)** du layer via un script, on ajuste l'intensité de chaque animation. Cela permet d'avoir des transitions fluides entre les animations, comme marcher tout en visant, tout en maintenant un contrôle précis sur chaque mouvement du personnage.

† Gestion d'IK (Inverse Kinematics) des pieds

Un script `FootControllerIK.cs` utilise l'inverse kinematics (IK) pour ajuster dynamiquement la position et l'angle des pieds en fonction du terrain. Il détecte la surface sous les pieds via des raycasts et met à jour leur placement dans `LateUpdate` et `OnAnimatorIK`, offrant une interaction naturelle et réaliste avec le sol pour un rendu visuel immersif.



Figure (3, 40) : Aperçu ON/OFF du Inverse Kinematics des pieds

† Utilisation du Package Animation Rigging

Pour renforcer le réalisme des interactions, les composants Multi-Aim Constraint et Two Bone IK Constraint du package Animation Rigging ont été utilisés. Le Multi-Aim Constraint oriente les bones vers une cible, avec limites pour éviter les déformations, facilitant le contact avec des objets. Le Two Bone IK Constraint assure le placement précis des armes, corigeant en temps réel les variations d'animation importées de Blender pour une meilleure stabilité lors des interactions avec les armes.

† Système de Ragdoll

Le système de **Ragdoll** dans Unity simule la physique des corps lorsqu'un personnage perd le contrôle, comme lors d'une chute ou d'une collision. Pour créer un Ragdoll, Unity permet de sélectionner les articulations du modèle 3D (bras, jambes, torse, tête) et d'ajouter automatiquement des composants **Rigidbody** et **Collider**.

Dans les scripts de contrôle des personnages, ennemis ou NPC, l'effet de Ragdoll est activé en désactivant l'**Animator** du personnage humanoïde. En exemple le script **BulletTarget** active l'effet Ragdoll en désactivant l'**Animator** et le **Collider** du personnage lorsqu'il est touché par une balle (BulletProjectile). Cela permet de remplacer l'animation par la physique réaliste lors de l'impact.

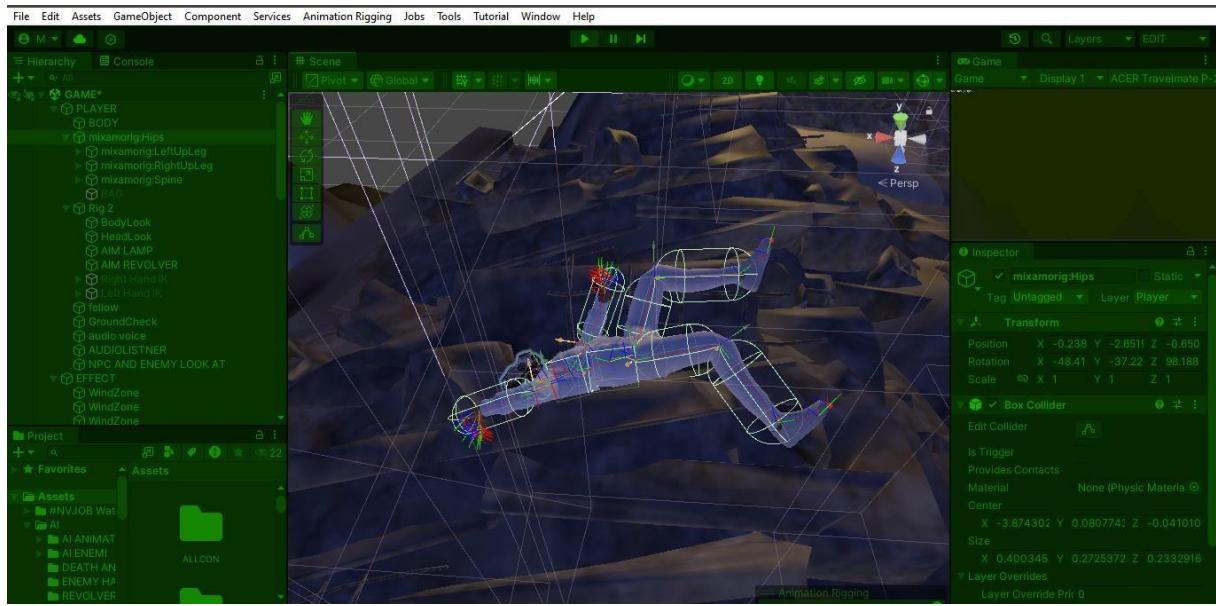


Figure (3, 41) : Visualisation de l'effet Ragdoll sur le joueur à l'état mort

† Cloth simulation

Le composant Cloth dans Unity est utilisé pour simuler des effets de textile réalistes en appliquant des comportements physiques, comme la gravité et le vent, aux objets de tissu. Le Cloth permet de contrôler la souplesse, la rigidité, et les interactions du tissu avec d'autres objets.

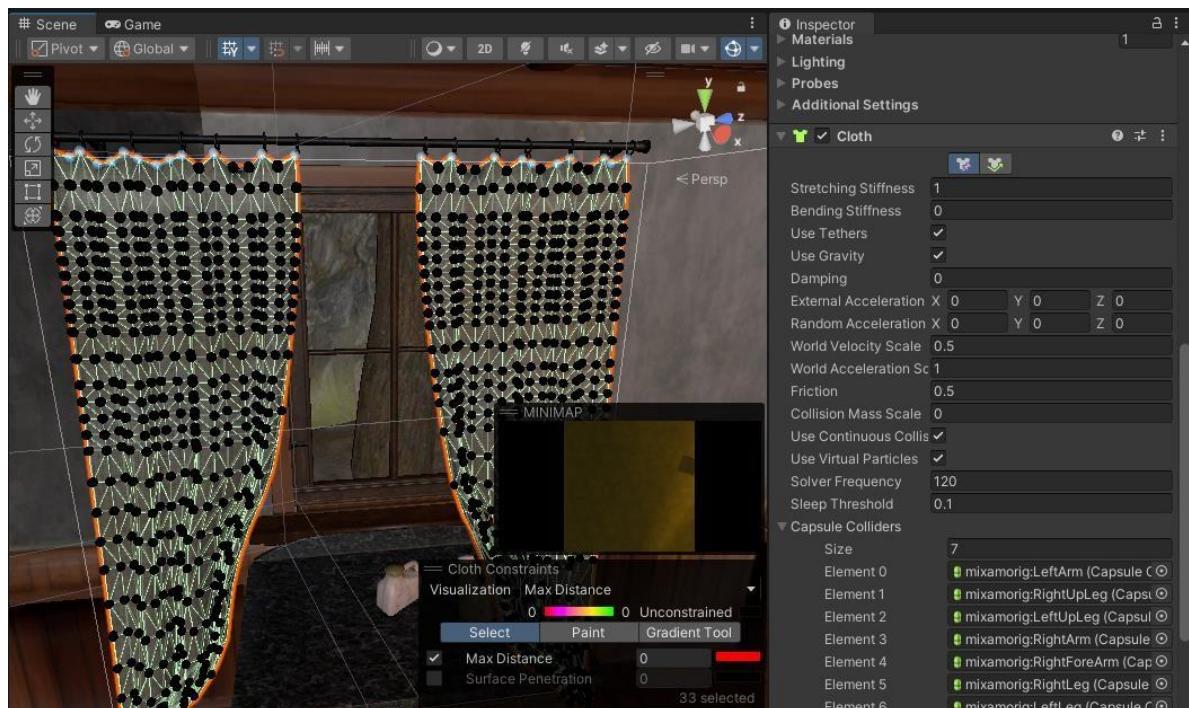


Figure (3, 42) Aperçu du composant Cloth sur un rideau

3.6.8 Fin de l'aventure

Pour conclure la scène GAME, un **BoxCollider** est placé de manière cachée à l'intérieur d'un rocher. À côté de ce BoxCollider, deux effets de **PostProcessingVolume** sont utilisés : le premier réduit la saturation des couleurs, tandis que le second augmente la luminosité. Lorsque le joueur entre dans cette zone et se cogne contre le **BoxCollider**, ce dernier déclenche le script, marquant ainsi la fin de la scène de jeu.

3.6.9 Réalisation de la scène ENDSTORY

Dans la scène **ENDSTORY**, la vidéo de fin d'histoire du personnage est jouée pour diffuser une vidéo de conclusion. Une fois la vidéo terminée, le jeu passe automatiquement à la scène CRÉDITS.

3.6.10 Réalisation de la scène CREDITS

Dans la scène Crédit, des informations sur le développeur défilent automatiquement grâce à une animation dans un UI ScrollView. L'animation de défilement n'est pas en boucle, et lorsqu'elle se termine, le script EndCredits.cs active une interface affichant le bouton « Quitter le jeu » pour permettre au joueur de fermer l'application.

Chapitre 4 : Défis et Solutions techniques

4.1 Problèmes rencontrés

Malgré les démarches d'optimisation dès le départ, le projet a rencontré des problèmes, notamment des bugs dans les builds finaux qui n'apparaissaient pas dans l'éditeur :

- une faible fluidité ou de fréquence d'image par seconde (6 à 8 fps)
- des bugs visuels où certaines parties des objets disparaissaient sous certains angles
 - des ombres et des lumières qui apparaissaient mal, des déformations, des artefacts visuels et des problèmes d'animation
- des temps de chargement élevés
- des erreurs fréquentes, telles que les **NullReferenceException**, ont également posé problème, empêchant certains scripts de fonctionner correctement
- l'interface utilisateur qui présentait des lags ou un mauvais affichage, notamment avec des layouts dynamiques mal configurés

- des animations qui ne se déclenchaient pas correctement
- des problèmes de blending ou d'IK engendraient des comportements visuels étranges
 - des objets qui traversaient d'autres objets
- des erreurs de rigidbodies qui provoquaient des mouvements inattendus
- des problèmes d'interface utilisateur mal dimensionnée ou mal positionnée, rendant le jeu difficile à utiliser sur différents appareils et résolutions.

4.2 Correction et optimisation des performances

Cette section détaille les méthodes utilisées pour résoudre les problèmes et améliorer les performances du jeu. Divers aspects ont été optimisés, incluant les graphismes, les scripts, l'audio, la physique et d'autres éléments, afin de garantir une expérience fluide sur différentes configurations matérielles.

4.2.1 Utilisation du Profiler et Stats

Le **Profiler** d'Unity a été essentiel pour identifier les problèmes de performance en visualisant l'impact de chaque composant (CPU, GPU, mémoire) et en détectant les goulets d'étranglement. Il a permis de repérer des anomalies comme des animations trop gourmandes ou des scripts inefficaces, facilitant ainsi l'optimisation ciblée. L'outil **Stats** complète le Profiler en offrant un aperçu rapide des performances, incluant les FPS et l'utilisation des ressources, idéal pour les tests en temps réel pendant le développement.



Figure (4, 1) : Fenêtre Stats

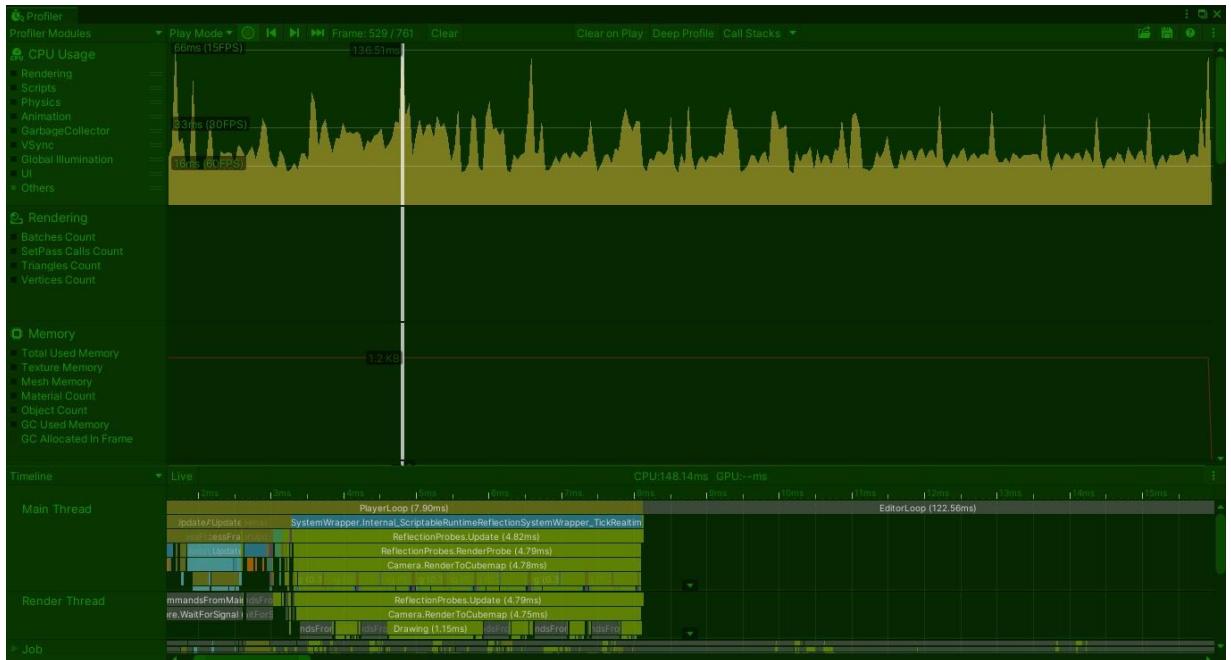


Figure (4, 2) : Profiler d'Unity

4.2.2 Debuging dans la fenêtre Console (Debug Log)

La Console, en complément du Profiler, est essentielle pour déboguer. Les messages Debug.Log permettent de vérifier les fonctionnalités, détecter les erreurs, et ajuster le code rapidement.

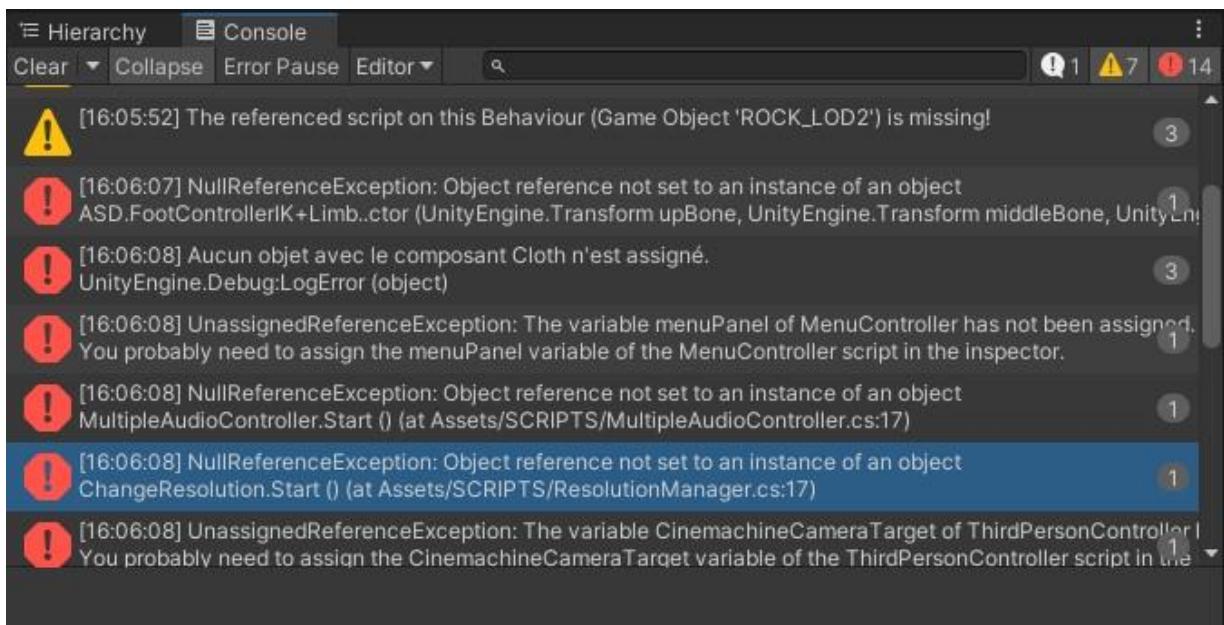


Figure (4, 3) : Fenêtre Console

4.2.3 Optimisation du graphique

Pour optimiser les performances graphiques, plusieurs composants et techniques ont été élaborés dans le but principal d'augmenter le nombre de FPS :

4.2.3.1 Composant LOD (Level of Detail) :

Le composant LOD Group (Level of Detail) a été ajouté pour ajuster automatiquement le niveau de détail des modèles 3D selon leur distance à la caméra. Dans Blender, chaque modèle a été simplifié avec le modificateur Decimate pour réduire les polygones, créant ainsi des versions allégées. Cependant, une réduction excessive peut créer des défauts visuels. Après Decimate, les outils Merge by Distance et Delete Loose ont permis de fusionner les sommets proches et de supprimer les éléments isolés, ce qui optimise la géométrie sans complexité superflue.

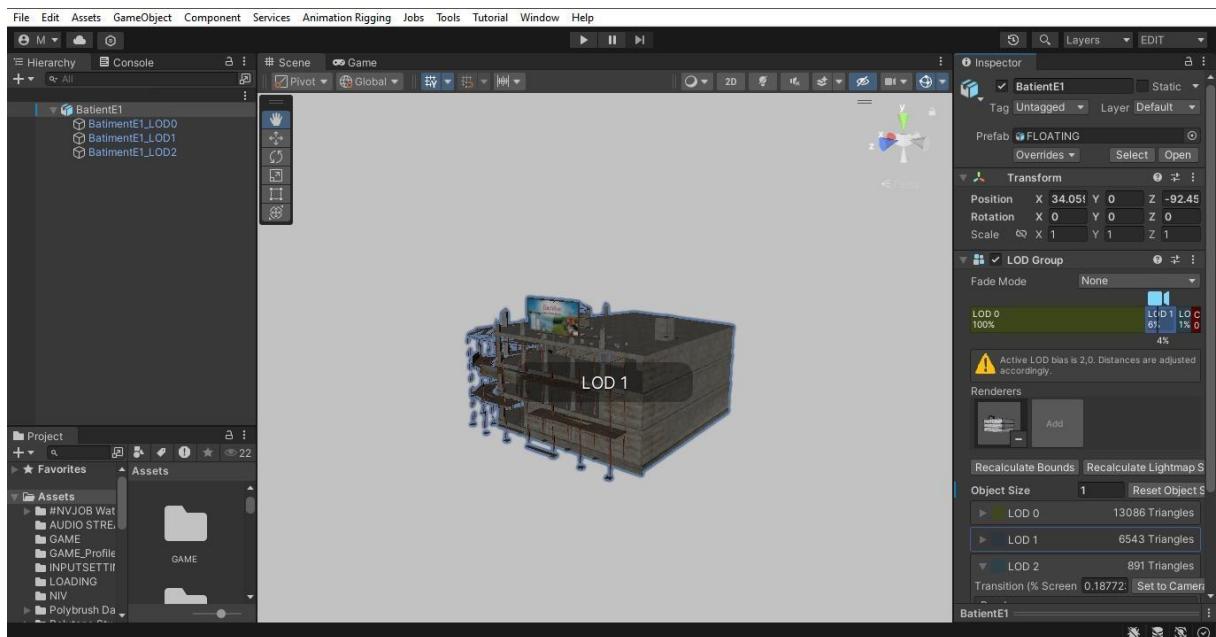


Figure (4, 4) : Composant LOD Group

Pour certains modèles, le modificateur Decimate étant insuffisant, une remodélisation a été nécessaire. Trois niveaux de LOD (LOD 0, LOD 1, LOD 2) ont été définis. Le niveau le plus bas (LOD 3) a requis 10 % de remodélisations, tandis que 90 % ont été optimisés avec Decimate. Chaque mesh dans Blender est renommé avec un suffixe (_LOD0, _LOD1, _LOD2) pour une attribution automatique des niveaux de détail dans Unity.

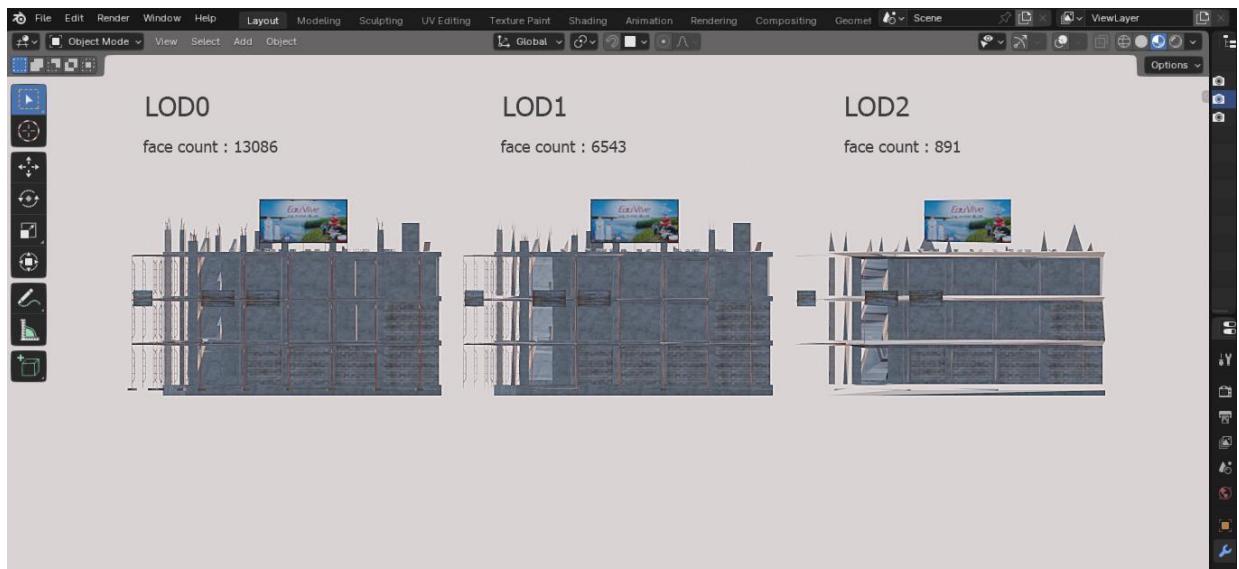


Figure (4, 5) : Présentation des niveaux de détail

4.2.3.2 Technique de Baking light

L'optimisation de l'éclairage a été réalisée grâce à l'utilisation du Baking Light, permettant de distinguer les éléments statiques des dynamiques. Les objets statiques ont été marqués comme tels dans Unity, ce qui a permis de pré-calculer l'éclairage via des **lightmaps**, réduisant ainsi les calculs en temps réel. Les objets dynamiques, tels que les personnages, armes et ennemis, ont continué à bénéficier d'un éclairage en temps réel. Le Baking Light a généré des lightmaps pour améliorer les performances sans sacrifier la qualité visuelle, tandis que des **probes** ont été utilisées pour assurer une interaction lumineuse correcte pour les objets mobiles.

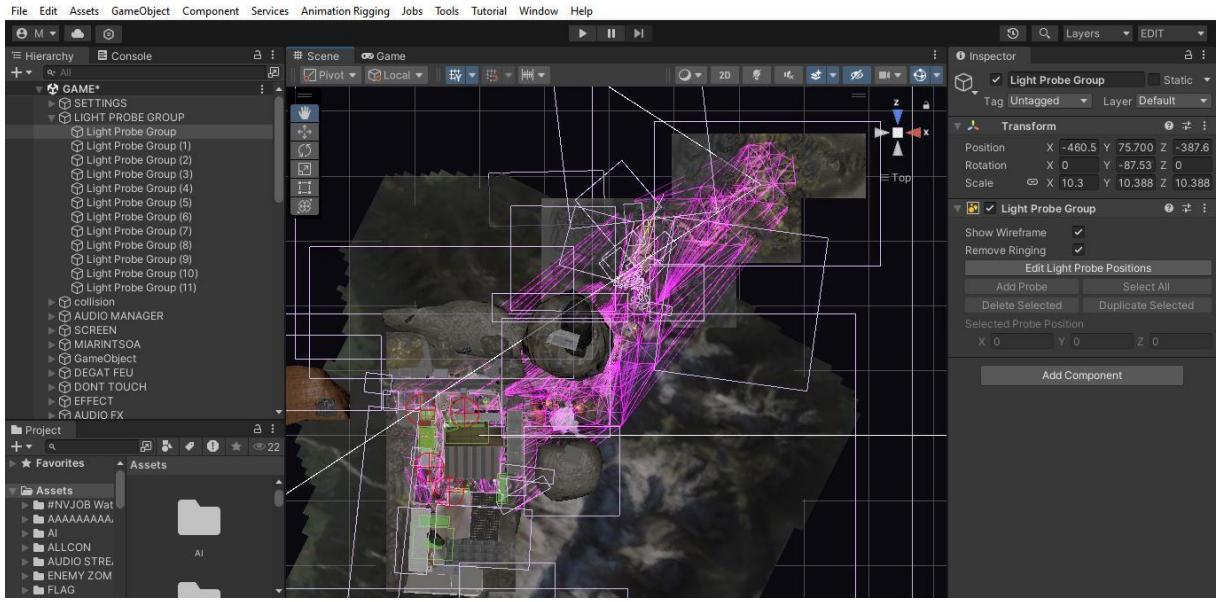


Figure (4, 6) : Visualisation du Light Probe Group

4.2.3.3 Utilisation de l'Occlusion Culling

L'Occlusion Culling est une technique d'optimisation qui réduit les calculs de rendu en cachant les objets non visibles à la caméra. Elle identifie les objets bloqués par d'autres (occluders) et évite de rendre ceux qui ne sont pas visibles (occludees). Les objets statiques ont été marqués pour permettre à Unity de pré-calculer les occlusions en fonction de la position de la caméra. Cette méthode améliore la performance globale en optimisant le rendu en temps réel.

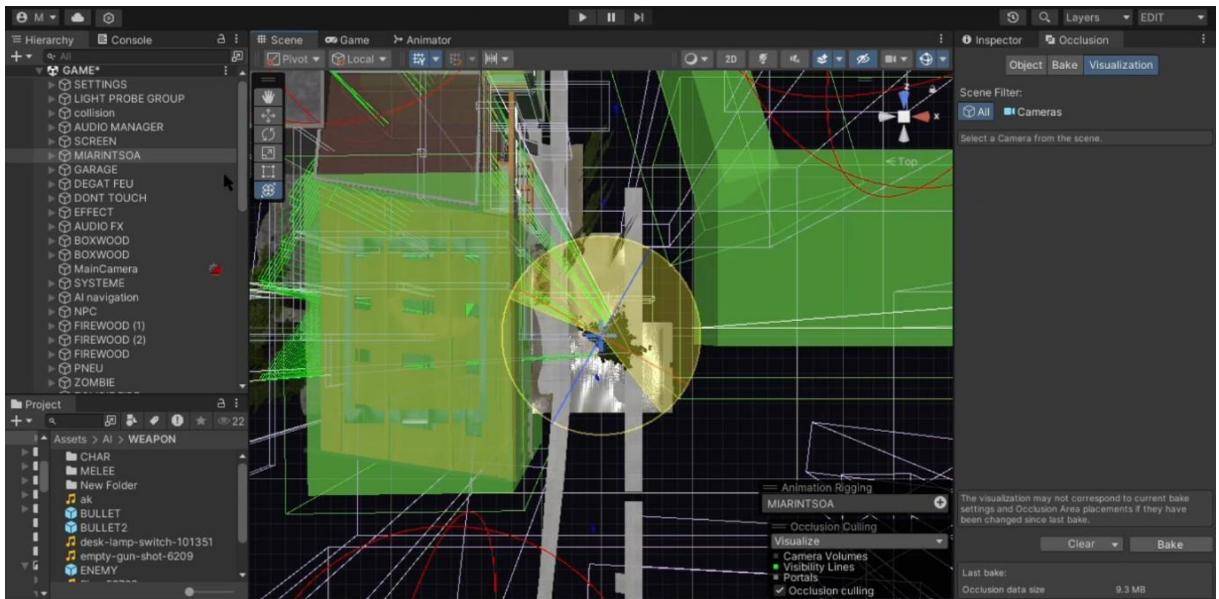


Figure (4, 7) : Visualisation de l'occlusion culling

4.2.3.4 Far clip plane

Le Far Clip Plane est un paramètre clé de la caméra dans Cinemachine, définissant la distance maximale à partir de laquelle les objets ne sont plus rendus. En limitant cette portée, il réduit le nombre d'objets affichés, optimisant ainsi l'utilisation des ressources GPU. Ce réglage est particulièrement utile dans les environnements ouverts, offrant un bon compromis entre qualité visuelle et performances.

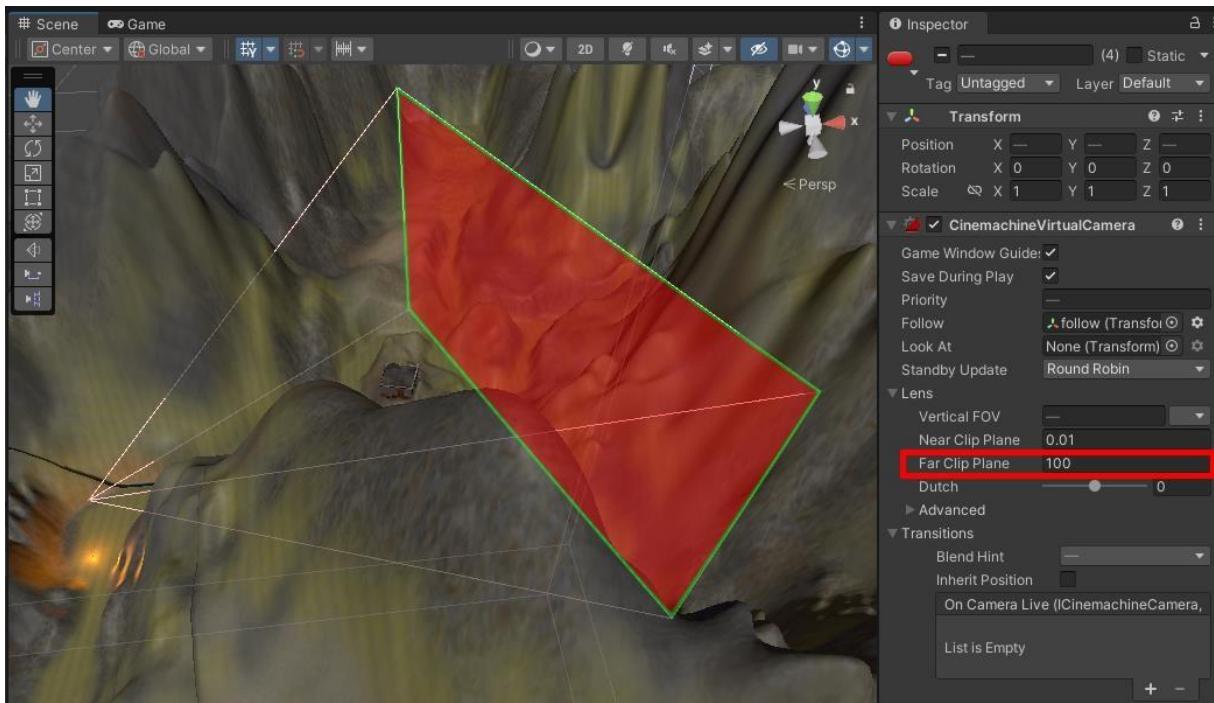


Figure (4, 8) : Visualisation du Far Clip Plane dans CinemachineVirtualCamera

4.2.3.5 Billboard Trees

Les Billboard Trees dans Unity optimisent les performances en remplaçant les modèles 3D d'arbres par des images planes (billboards) orientées vers la caméra. Lorsque les arbres sont éloignés, ces images 2D prennent le relais, réduisant la charge sur le GPU et améliorant les performances, notamment dans les environnements forestiers denses, tout en conservant une qualité visuelle acceptable.

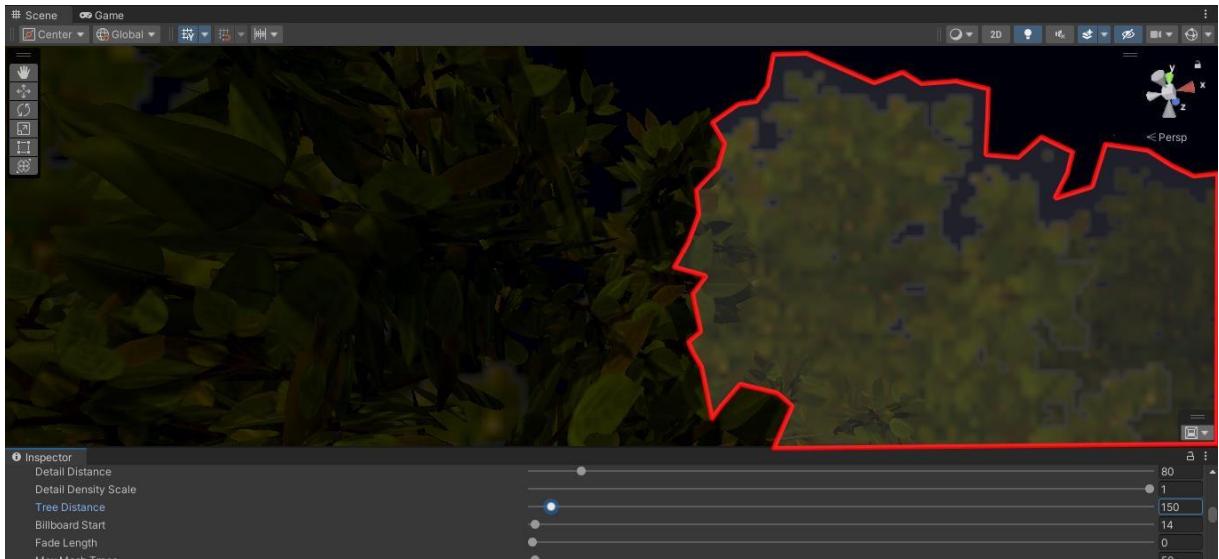


Figure (4, 9) : Visualisation du Billboard Tree

4.2.3.6 Création de modèles Low Poly et de surfaces planes

Réduire le nombre de polygones pour les objets secondaires ou éloignés est une technique d'optimisation efficace, diminuant la charge de rendu et libérant des ressources pour les éléments principaux du jeu. L'utilisation de sprites 2D sur des planes pour les arrière-plans éloignés est également une méthode visuelle performante, réduisant considérablement la charge de calcul. Ces approches permettent d'améliorer les performances sans nuire à la qualité visuelle du jeu, notamment dans des scènes complexes.

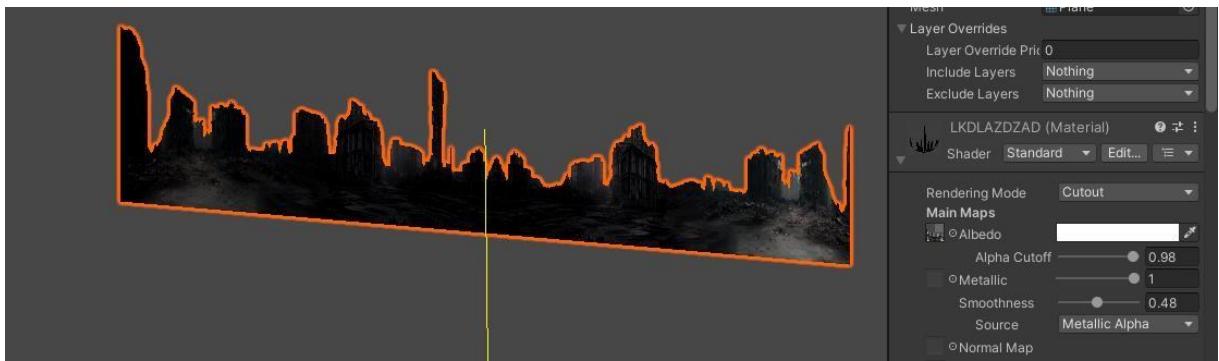


Figure (4, 10) : Plane avec texture en mode Cutout

4.2.3.7 Réduction de la Résolution des Textures

Réduire la résolution des textures dans Photoshop, par exemple en passant de 4K (4096 x 4096) à 1K (1024 x 1024) ou 512 pixels pour les objets secondaires, permet d'alléger la charge sur le GPU. Cette optimisation réduit l'utilisation de la mémoire vidéo (VRAM), accélère les temps de chargement et améliore le framerate global du jeu.

4.2.3.8 Technnnique du Batching

Le batching dans Unity optimise les performances en combinant plusieurs objets similaires pour les rendre en une seule opération de dessin. Voici les principaux types de batching :

Static Batching : Combine les objets statiques (non mobiles), réduisant les draw calls (appel de rendu) pour des scènes avec beaucoup d'objets fixes. Cependant, il nécessite plus de mémoire pour stocker le mesh combiné.

Dynamic Batching : Combine les objets non statiques avec un faible nombre de vertex (moins de 900), comme les petits objets en mouvement partageant le même matériau. Ce type de batching est limité aux objets simples, sans effets de lumière complexes.

GPU Instancing : Permet d'envoyer plusieurs copies d'un objet identique au GPU avec une seule draw call. Idéal pour des objets répétés (arbres) en mouvement, mais nécessite des shaders compatibles et des cartes graphiques modernes.

Ces méthodes de batching permettent de réduire les draw calls.

4.2.3.9 Optimisation des Shaders

Les shaders complexes peuvent impacter les performances du jeu. Pour optimiser le rendu, l'utilisation de shaders simples, tels que Mobile/Diffuse ou Unlit, réduit la consommation de ressources, tandis que l'application d'un même matériau sur plusieurs objets limite les changements de matériaux, améliorant ainsi la performance.

4.2.3.10 Réduction des Effets

Les effets de post-processing (comme le bloom, le flou de mouvement ou l'occlusion ambiante) peuvent être très gourmands. En les appliquant uniquement lorsque nécessaire (par exemple, dans des cinématiques ou des scènes intenses), les performances du jeu en temps réel sont améliorées.

4.2.3.11 Optimisation du composant Particle System

L'optimisation des systèmes de particules dans Unity inclut la réduction du nombre de particules, l'utilisation du LOD pour simplifier les effets à distance, et l'application de textures de faible résolution. Il est aussi essentiel de limiter les effets de post-traitement coûteux, d'activer l'instancing GPU pour réduire les appels de rendu, et de limiter les collisions et forces. Désactiver les particules hors écran améliore la fluidité et les performances tout en maintenant la qualité visuelle.

4.2.3.12 Utilisation des faces triangulaires

Des problèmes d'animation ont été constatés en raison de l'utilisation de quads dans Blender, non pris en charge par Unity, entraînant des déformations visuelles. Pour y remédier, les quads ont été convertis en triangles à l'aide de la fonction "Triangulate Faces" dans Blender, garantissant ainsi une meilleure compatibilité avec Unity.

4.2.4 Corrections des animations

Des problèmes d'animation ont été rencontrés lors des pauses du jeu ou lors de transitions entre les scènes. Pour y remédier, la propriété Time.unscaledDeltaTime a été utilisée, permettant de maintenir la fluidité des animations de l'UI pendant les pauses et de garantir leur bon déclenchement lors des changements de scène, assurant ainsi une expérience utilisateur cohérente.

4.2.5 Corrections des boutons et mise en place du responsive UI

4.2.5.1 Priorité des Boutons

Les priorités dans l'outil d'UI ont été ajustées pour garantir que les boutons essentiels soient correctement affichés et accessibles.

4.2.5.2 Utilisation du composant Canvas Scaler

Pour assurer une bonne responsivité, le Canvas Scaler a été utilisé dans Unity. Voici les principales étapes mises en place :

Mode de mise à l'échelle : Configuration en mode "Scale with Screen Size" pour ajuster automatiquement l'échelle de l'UI en fonction de la taille de l'écran.

Match : Le paramètre "Match" a été ajusté pour équilibrer la mise à l'échelle entre la largeur et la hauteur de l'écran.

Références de résolution : Une résolution de référence (1366x768) a été définie pour garantir une expérience cohérente sur différents appareils.



Figure (4, 11) : Composant Canvas Scaler

4.2.5.3 Utilisation des composants Layouts adaptives et RectTransform

Pour garantir l'accessibilité et l'esthétique de l'interface utilisateur sur différents appareils, des layouts adaptatifs ont été utilisés :

Vertical Layout Group : Organise les éléments verticalement avec un espace automatique, idéal pour les menus et listes

Horizontal Layout Group : Aligne les éléments horizontalement, adapté aux barres de navigation ou options en ligne

Layout Element : Permet de contrôler la taille et l'espacement des éléments, offrant une personnalisation fine

Ancrage et Pivot : En configurant correctement les points d'ancrage et de pivot, les éléments s'ajustent de manière cohérente lors du redimensionnement de la fenêtre

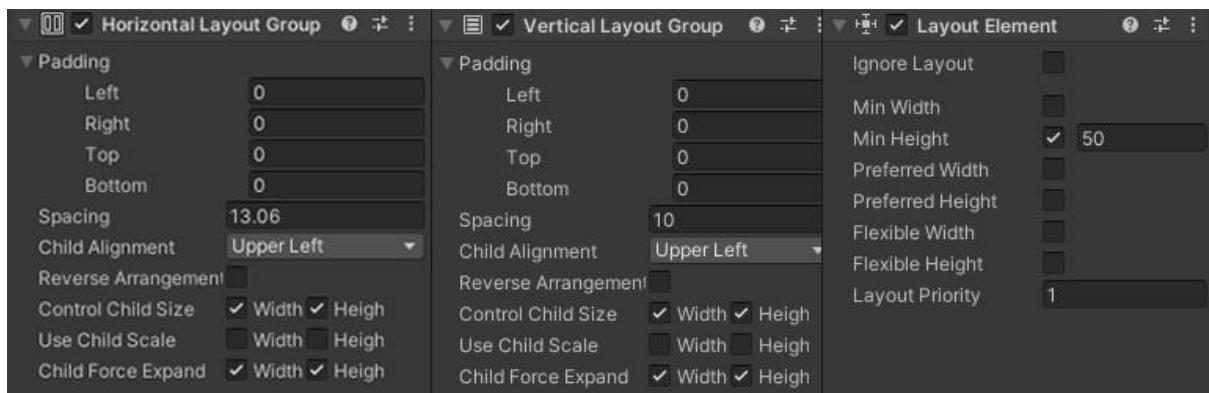


Figure (4, 12) : Composant Layout

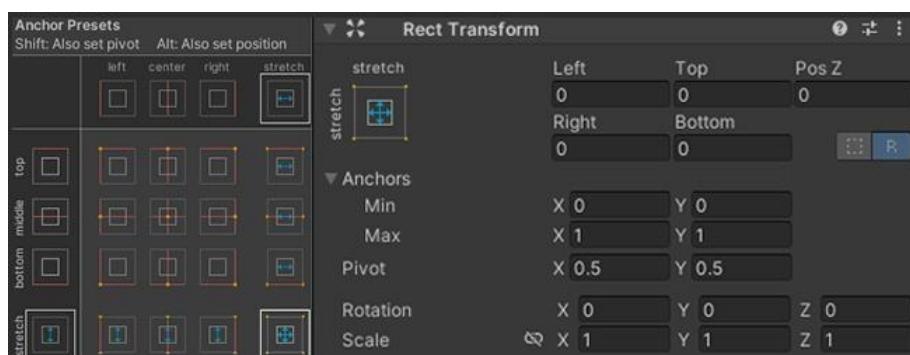


Figure (4, 13) : Composant Rect Transform

4.2.6 Optimisation des scripts

Minimisation des appels à Update() : La fonction Update() a été réduite en déplaçant les calculs lourds vers des fonctions appelées moins fréquemment, comme FixedUpdate().

Utilisation de coroutines : Des coroutines ont été utilisées pour exécuter des tâches en arrière-plan de manière plus efficace, réduisant les ralentissements.

4.2.7 Solution du Backface Culling

Le backface culling est une technique qui exclut l'affichage des faces arrière des objets 3D, optimisant ainsi le rendu graphique. Dans ce projet, une vérification de l'orientation des faces a été effectuée dans Unity. En cas d'incohérence, des corrections ont été réalisées dans Blender, en utilisant l'option de retournement des faces.

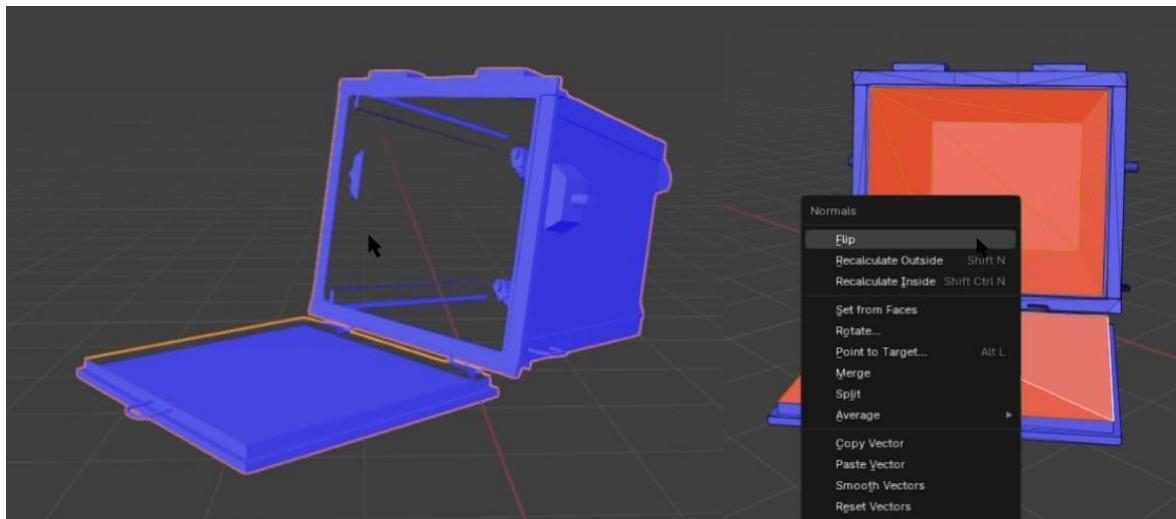


Figure (4, 14) Visualisation et face orientation du problème du backface culling

4.2.8 Optimisation Audio

La compression audio a été effectuée en utilisant des formats optimisés tels que **MP3**, réduisant ainsi la consommation de mémoire et les temps de chargement, sans perte significative de qualité sonore. Le **streaming audio** a été mis en place pour les fichiers volumineux, comme les musiques de fond, afin d'éviter la surcharge de mémoire et d'optimiser les performances.

4.2.9 Optimisation physique et mémoire

4.2.9.1 Modèle Simplifié pour les Mesh Colliders

Le MeshCollider dans Unity permet des collisions précises mais est plus coûteux en ressources. Pour l'optimiser, les objets ont été simplifiés dans Blender pour créer des modèles adaptés, réduisant la complexité des calculs. Une alternative consiste à utiliser plusieurs box colliders pour les parties essentielles, bien que cela nécessite plus de configuration.

4.2.9.2 Désactivation des physiques via un script

Plusieurs scripts ont été créés pour gérer différentes fonctionnalités, notamment pour désactiver certains éléments lorsque le joueur ne se trouve pas dans un BoxCollider. Cela inclut des éléments comme les systèmes de particules, les Rigidbody, les effets miroir en temps réel avec Reflection Probe, les animations, les simulations de tissu (cloth), les mouvements aléatoires des IA et d'autres effets visuels. Par exemple, le script ClothActivator.cs désactive le composant Cloth au départ afin d'économiser des ressources. Lorsque le joueur entre dans une zone, le tissu est activé, et lorsqu'il en sort, il est désactivé. Cette approche permet d'optimiser les performances en réduisant les calculs inutiles liés à la simulation du tissu.

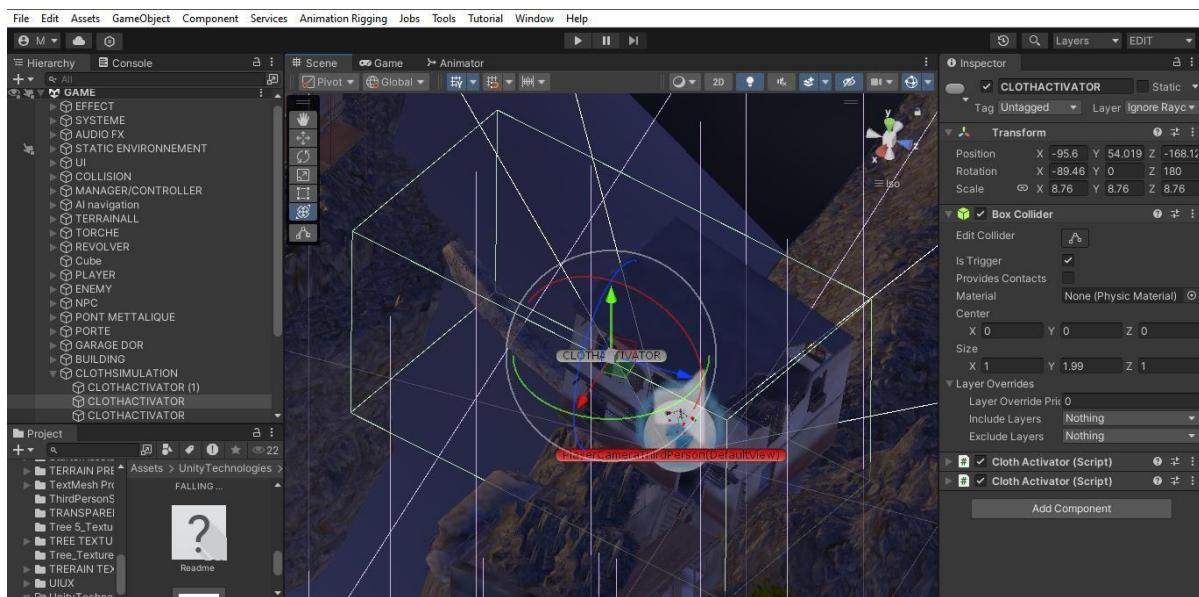


Figure (4, 15) : Visualisation de la zone d'activation du composant Cloth

4.2.10 Optimisation du NavMesh

Pour optimiser le système de navigation, des **NavMesh Modifier Volumes** ont été utilisés pour délimiter des zones spécifiques qui ne doivent pas être prises en compte par le système de navigation. Cela permet d'éviter des recalculs inutiles, surtout dans un terrain de grande taille, en excluant les zones non pertinentes. L'**Area Type** a été configuré sur **NotWalkable** pour ces volumes afin d'empêcher la navigation dans ces espaces et ainsi réduire les coûts de calcul.

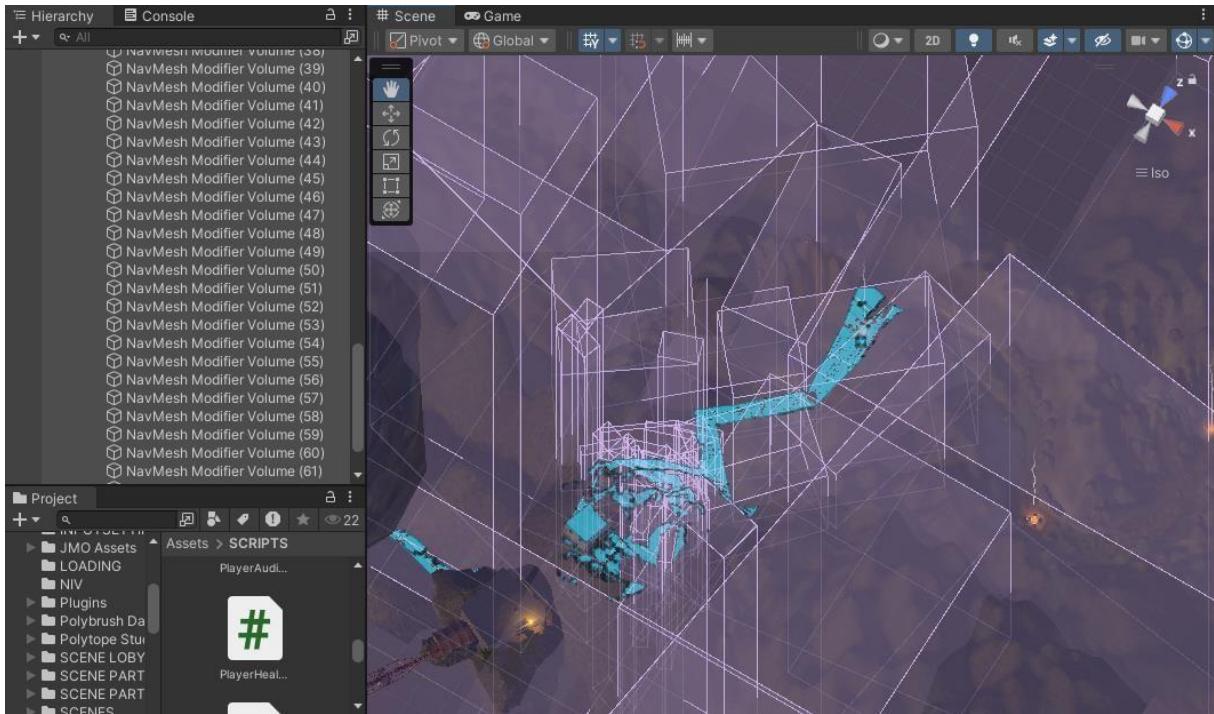


Figure (4, 16) : Visualisation du NavMesh optimisé

4.2.11 Option Paramètre Graphique

Dans la section dédiée à l'optimisation du jeu, un aspect essentiel a été d'offrir au joueur la possibilité de personnaliser les paramètres graphiques selon ses préférences. Une section dédiée aux paramètres graphiques a donc été intégrée, comprenant plusieurs scripts :

- **ChangeQuality.cs** : Ce script permet au joueur d'ajuster la **qualité graphique globale** via un menu déroulant (TMP_Dropdown), en sélectionnant l'un des niveaux prédéfinis dans Unity.
- **ResolutionManager.cs** : Ce script gère la **Résolution d'écran** et le mode plein écran, permettant au joueur de choisir une résolution adaptée parmi les options disponibles et de basculer entre plein écran et fenêtre.
- **PostProcessingManager.cs** : Ce script permet d'activer ou désactiver des **effets visuels** tels que le bloom, la profondeur de champ, les réflexions, et autres effets graphiques avancés, grâce à des boutons basculants.

En plus de ces options, d'autres paramètres tels que **HDR (High Dynamic Range)**, **MSAA (Multisample Anti-Aliasing)**, les **ombres** et **l'éclairage en temps réel** ont également été ajoutés, offrant au joueur un contrôle total sur les réglages graphiques. Ces outils permettent non seulement une flexibilité optimale, mais également une **amélioration significative des**

performances, en ajustant les réglages graphiques en fonction des capacités matérielles du système utilisé.

4.2.12 Corrections des Collisions

Des box colliders ont été ajoutés par-dessus les mesh colliders pour simplifier les surfaces de collision et éviter que le personnage ne se coince dans les angles ou les arêtes. Cette approche permet des déplacements plus fluides tout en limitant les zones de friction sans perdre en précision de collision.

4.2.13 Gestion des assets non utilisés :

L'utilisation de l'**Asset Usage Detector** (voir Annexe 4) (disponible dans Asset Store) a permis une gestion optimisée des ressources en identifiant les assets non utilisés, réduisant ainsi la taille du projet et améliorant les performances globales. Cet outil a également facilité l'organisation des ressources et augmenté la flexibilité du développement, contribuant à une allocation de mémoire plus efficace et à une meilleure qualité de l'expérience de jeu.

4.2.14 Chargement asynchrone des scènes

Pour optimiser les transitions entre les scènes dans Unity, un système de chargement asynchrone a été mis en place en utilisant deux scripts et des canvas de transition. Le script principal, `ActivateAndDeactivateGameObject.cs`, active un canvas de finalisation au début de chaque scène, puis le désactive après un délai, offrant un affichage temporaire pour des éléments de chargement.

Un second script, intégré partiellement dans d'autres scripts, gère le chargement asynchrone avec un canvas de transition, qui inclut une réduction progressive du volume de la musique et un écran de chargement. Pour éviter les effets de latence visuelle, la caméra principale utilise un fond noir pour masquer la scène vide, et les vidéos intégrées ont été conçues avec un fond noir pour assurer une transition fluide.

Chapitre 5 : Discussion et résultats attendu

Les objectifs principaux du projet ont été atteints, malgré plusieurs défis techniques, notamment des crashes nécessitant des réimportations du projet. Cependant, des points restent à améliorer, notamment au niveau du système de sauvegarde. Bien que celui-ci ait été testé, le grand nombre d'objets présents dans la scène rend difficile la création d'une sauvegarde

fiable pour chaque état sans causer de problèmes sur d'autres états. Cela est particulièrement vrai pour les animations, qui combinent des animations brutes et procédurales. Cette complexité ajoute des défis lors de la gestion des états et de leur restauration, nécessitant des ajustements supplémentaires pour garantir une sauvegarde efficace et fonctionnelle.

5.1 Défis Techniques et Résolutions

Le développement du jeu a rencontré plusieurs défis techniques, au-delà des problèmes d'optimisation, qui ont nécessité une réflexion approfondie et des ajustements constants. L'un des premiers problèmes rencontrés a été une coupure de courant, provoquant une défaillance des composants et rendant l'ouverture du projet impossible. Cela a nécessité une réimportation manuelle de tous les éléments pour identifier celui qui avait échoué. Après ce problème, une solution a été mise en place : la création d'un package en local ou en ligne contenant tous les éléments, similaire à ceux disponibles dans l'Asset Store.

Cela permet désormais de prévenir les incidents similaires.

Un autre défi concernait la gestion des éléments du projet, notamment à cause de la renommée des fichiers. Au lieu de renommer correctement les éléments pour faciliter leur identification, ils ont été renommés de manière aléatoire (ex. : daffahfzaf.jpg), ce qui a créé de la confusion. Pour éviter ce genre de situation, chaque élément doit avoir un nom significatif. De plus, un outil de détection a été utilisé pour rechercher où chaque élément est utilisé en cas de confusion.

Concernant la gestion du travail, l'utilisation des layouts de fenêtres a été très utile pour organiser le travail dans tous les logiciels. Pendant l'optimisation du jeu, le processus de **baking light** a été particulièrement frustrant, car lors du premier test, l'estimation du temps pour cette tâche était de 39 jours. Cependant, en simplifiant les objets et en retirant certains éléments, le baking a pu être réalisé en 13 jours.

Un problème lié à la gestion du temps est survenu avec l'utilisation de variables privées dans le code pour des raisons de sécurité. Cela a causé une perte de temps, car ces variables n'étaient pas modifiables directement dans l'inspecteur, nécessitant une recompilation du code à chaque modification. La solution a été de mettre ces variables privées dans des SerializedField, ce qui permet de les modifier dans l'inspecteur sans avoir besoin de recompilation à chaque fois.

Un autre défi majeur a été la gestion de la modularité du code, notamment en ce qui concerne les multiples scripts destinés à contrôler le mouvement du joueur et ses interactions avec l'environnement. Des conflits entre certaines classes ont émergé, nécessitant une réorganisation du code afin de renforcer la modularité et d'éviter les problèmes d'intégration.

Ce processus a permis de simplifier l'ajout de nouvelles fonctionnalités tout en maintenant la cohérence du projet.

5.2 Tests et Validation

Afin de garantir la stabilité et la performance du jeu, une série de tests rigoureux a été réalisée. Des sessions de tests de gameplay ont permis d'évaluer la réactivité des contrôles, la fluidité des animations et l'équilibrage des mécaniques de jeu. L'utilisation de l'outil Profiler de Unity a été indispensable pour analyser la charge CPU et la consommation mémoire, permettant de repérer les pics de performances et d'implémenter des solutions d'optimisation. Cette approche a permis de garantir une fluidité optimale, même lors de scènes complexes ou en présence de nombreux objets interactifs.

Des tests de compatibilité ont également été menés sur diverses configurations matérielles pour assurer que le jeu fonctionne efficacement sur une gamme étendue de plateformes. Cela a permis de détecter et de corriger certains problèmes liés à la performance sur des configurations plus anciennes ou moins puissantes, assurant ainsi une expérience de jeu homogène pour tous les utilisateurs.

5.3 Synthèse des Acquis et Perspectives

Ce projet a permis de développer une expertise approfondie dans la conception et la réalisation d'un jeu vidéo, couvrant l'ensemble du processus de production, de la conception initiale à la finalisation. La gestion du projet, avec une coordination efficace entre les différentes étapes de développement, a permis de surmonter les défis techniques et de garantir la cohésion entre les éléments créatifs et techniques. Cette expérience a également permis d'affiner des compétences clés telles que la modélisation 3D, la gestion des animations, l'intégration des effets visuels et la gestion des ressources sonores.

Les compétences acquises durant ce projet permettent désormais de gérer l'intégralité du processus de création d'un jeu vidéo, de la conception à la mise en œuvre, avec un accent particulier sur l'optimisation des performances et l'amélioration continue de la qualité du gameplay. Toutefois, certains aspects du projet auraient pu être approfondis. Par exemple, l'intégration de systèmes de physique plus complexes pour améliorer les interactions avec l'environnement ou la mise en place d'un système d'intelligence artificielle plus sophistiqué pour les ennemis.

Dans une version future du jeu, il serait pertinent d'explorer l'ajout de nouveaux types d'interactions, ainsi que des mécaniques de gameplay inédites, pour enrichir l'expérience utilisateur. La prise en compte des retours des joueurs lors de cette phase de test sera également essentielle pour apporter les ajustements nécessaires et maximiser l'engagement et l'immersion dans le jeu.

5.4 Perspectives d'Évolution

Pour enrichir davantage le projet, plusieurs axes d'évolution peuvent être envisagés, visant à approfondir et diversifier les compétences techniques et artistiques.

Exploration audio : Actuellement, la majorité des audios intégrés au jeu proviennent de ressources téléchargées. Une amélioration notable consisterait à créer des compositions originales, notamment des bandes sonores orchestrales adaptées à l'ambiance du jeu. Ce type d'audio renforcerait l'immersion et permettrait un contrôle plus précis sur l'accompagnement musical.

Utilisation de la capture de mouvements : Le recours à des technologies de capture de mouvements, telles que le système Rococo, offrirait la possibilité de produire des animations plus naturelles et complexes. Contrairement aux méthodes actuelles utilisées pour les animations, la capture de mouvements permettrait une plus grande fluidité et des interactions plus réalistes des personnages dans des situations variées.



Création de VFX (CGI) pour les vidéos : Dans le projet actuel, les génériques et autres séquences vidéo reposent sur des enregistrements réels. Une perspective d'évolution consisterait à approfondir les techniques de création CGI pour remplacer ou enrichir ces vidéos par des animations numériques plus immersives.

Amélioration des graphismes comme le HDRP qui donnerait une plus grande possibilité d'améliorer encore plus la qualité visuelle.

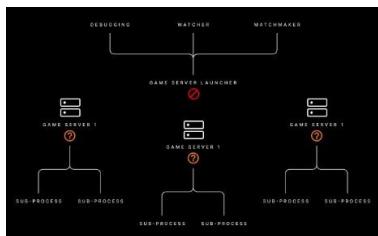


Ajout d'intelligence artificielle Machine Learning pour plus de complexité pour les ennemis, tels que la recherche dynamique du joueur ou des actions de groupe.



AI / ML integration

Multijoueur en ligne : En intégrant un système de jeu en réseau, le jeu pourrait évoluer vers une expérience multijoueur, ce qui augmenterait son intérêt et sa longévité.

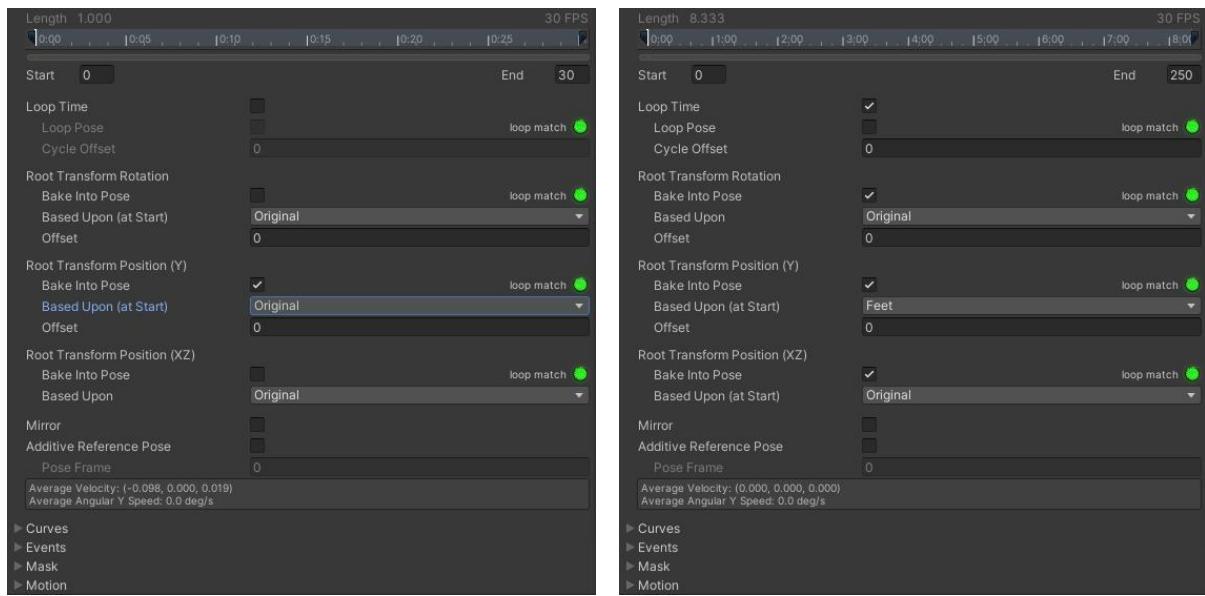


Optimisation avancée : Des techniques comme le multi-threading ou le pooling avancé d'objets pourraient également être mises en place pour optimiser la performance du jeu, en particulier sur les scènes les plus complexes.

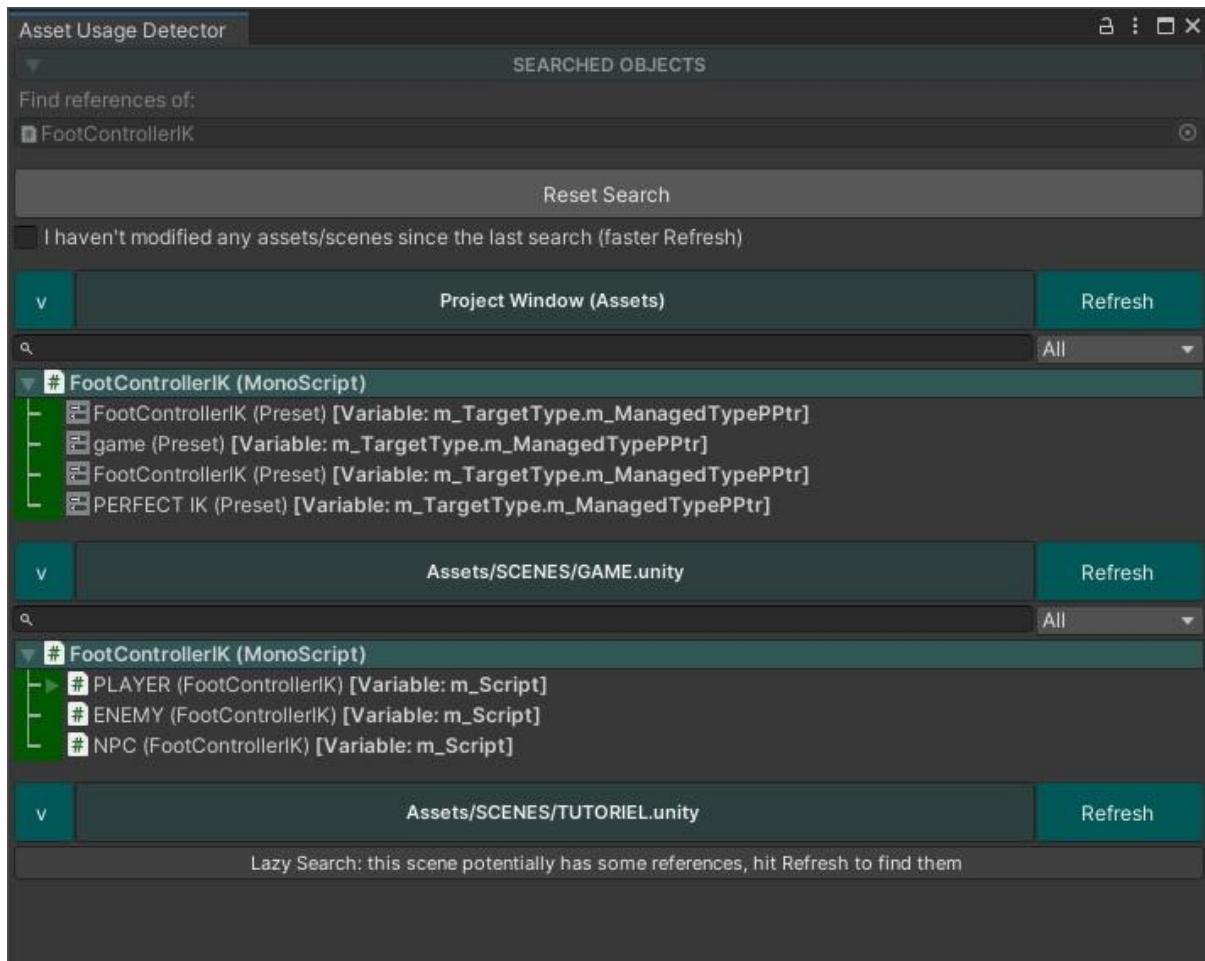
Caractéristiques	Unity 2020	Unity 2021	Unity 2022	Unity 2023	Unity 2024 (Prévision)
Lancement	2020	2021	2022 (LTS)	2023	2024 (à venir)
Type	Standard	Standard	LTS	Standard	Standard
Pipeline de rendu	URP introduit, HDRP en développement	Améliorations de HDRP et URP	Optimisations URP/HDRP	Améliorations continues	Nouvelles technologies graphiques
Scripting visuel	Introduction du Visual Scripting	Améliorations des outils visuels	Support complet pour Visual Scripting	Améliorations à la facilité d'utilisation	Attente d'outils avancés
Performance	Améliorations du moteur physique	Performances améliorées	Améliorations significatives	Optimisations supplémentaires	Nouvelles optimisations
Support XR	Support de base pour XR	Meilleures fonctionnalités XR	Support XR amélioré	Support XR renforcé	Attente de nouvelles fonctionnalités XR
Outils de collaboration	Outils de collaboration limités	Meilleure intégration avec Collaborate	Outils de collaboration avancés	Outils de collaboration améliorés	Prévisions d'améliorations
Communauté et ressources	Émergence de la communauté	Croissance de la base d'utilisateurs	Large communauté avec ressources robustes	Communauté en expansion	Nouvelle base d'utilisateurs

Annexe 2

Annexe 3



Annexe 4



I. BIBLIOGRAPHIE/WEBOGRAPHIE

DOCCUMENTATION

Titre	Lien
Learn Unity	https://unity.com/fr/learn
Unity User Manual 2022.3 (LTS)	https://docs.unity3d.com/Manual/index.html
Blender 4.0	https://docs.blender.org/manual/en/4.0/
C# Programming Guide	https://learn.microsoft.com/fr-fr/dotnet/csharp/
AI Navigation	https://docs.unity3d.com/Packages/com.unity.ai.navigation@1.1/manual/index.html
Animation Rigging	https://docs.unity3d.com/Packages/com.unity.animation.rigging@1.3/manual/index.html
Cinemachine	https://docs.unity3d.com/Packages/com.unity.cinemachine@2.3/manual/CinemachineBrainProperties.html
Polybrush	https://docs.unity3d.com/Packages/com.unity.polybrush@1.1/manual/index.html
FL Studio Reference Manual	https://www.image-line.com/fl-studio-learning/fl-studioonline-manual/

TUTORIELS YOUTUBE

Titre	Lien
Blender 4.0 Beginner Donut Tutorial (NEW)Blender Guru	https://youtube.com/playlist?list=PLjEaoINr3zgEPv5y--4MKpciLaoQYZB1Z&si=vcn-T_cVBBwe46ki
Animated Character Creation in Blender 3DCrossMind Studio	https://youtube.com/playlist?list=PLgO2ChD7acqFl3vtixa673FwCf2OfcCqK&si=KfR0VHPfnpC0blsk

Sites de téléchargements

Type	Lien
3D	https://sketchfab.com/feed
PBR Texture	https://ambientcg.com/
Vidéo/Audio	https://www.youtube.com/
Audio	https://pixabay.com/fr/
Icon	https://game-icons.net/
Package externe Unity	https://assetstore.unity.com/

Musique

Titre	Compositeur
♪ The Last of Us Part 1	Gustavo Santaolalla
♪ Can You Hear The Music	Ludwig Göransson
♪ Jacob and the Stone	Emile Mosseri
♪ Fourth of July	Sufjan Stevens