

Rapport Soutenance Groupe 30

Axel Gerifaud
Alex Barriol
Antoine Berier
Martin Lemetais

06 Mars 2023



Table des matières

1	Introduction	3
1.1	Le groupe	3
1.2	Répartition des tâches	4
2	Avancement du projet	4
2.1	Environnement	4
2.2	Ennemis	5
2.2.1	Les fondamentaux de l'intelligence artificielle	5
2.2.2	Pathfinding/Mesh Agent	5
2.2.3	Déplacement aléatoire	5
2.2.4	Animations	6
2.3	Armes	6
2.4	Gameplay	6
2.4.1	Déplacements	6
2.4.2	Joueur	7
2.4.3	Respawn	7
2.4.4	Système de tir	8
2.4.5	Interface	8
2.5	Multijoueur	8
2.6	Site Web	9
3	Planning prévision	10
3.1	Prochaine soutenance	10
4	Conclusion	11
5	Webographie	11
6	Annexes	12

1 Introduction

Nous sommes ravis de vous présenter notre premier rapport de soutenance pour notre jeu de tir à la première personne (FPS). Notre jeu a été conçu pour offrir une expérience de jeu immersive et engageante, en utilisant les dernières technologies pour créer un monde de jeu réaliste et fascinant. Notre équipe de développement a travaillé avec détermination pour développer des mécaniques de jeu intuitives, des personnages travaillés, des environnements détaillés et une histoire captivante.

Dans ce rapport de soutenance, nous allons présenter les détails de notre projet, en mettant l'accent sur les choix de conception, les fonctionnalités clés et les résultats obtenus. Nous allons également aborder les défis que nous avons rencontrés au cours du processus de développement et des solutions que nous avons mises en place pour y faire face. Enfin, nous allons planifier l'avenir de notre projet et les objectifs que nous avons pour son développement.

Nous espérons que ce rapport de soutenance vous donnera un aperçu détaillé de notre travail sur le jeu de tir à la première personne, ainsi que de notre approche du développement du jeu. Nous sommes impatients de partager notre travail et de recueillir vos commentaires et suggestions.

1.1 Le groupe

Nous sommes un groupe de 4 personnes nommé "JeuVONA" et nous travaillons bien ensemble sur ce projet, avec une bonne cohésion de groupe.

1.2 Répartition des tâches

Répartition des tâches	Alex	Axel	Martin	Antoine
Interface			R	S
Logo		S	R	
Son			S	R
Gameplay	R	R	R	R
Animation	R			S
Game Design	R	S		
Modélisation	S			R
IA	R	S		
Multijoueur	S	R		
Site Web			R	

2 Avancement du projet

2.1 Environnement

Imaginer et créer la map est un défi intéressant pour notre équipe, car nous avons une carte basique avec très peu d'éléments dessus. Cependant, nous avons vu cela comme une opportunité pour laisser libre cours à notre créativité et expérimenter de nouvelles idées. Nous avons commencé par établir une liste d'objectifs pour notre map, en nous assurant que nous offrons aux joueurs suffisamment de défis pour maintenir leur intérêt tout en évitant de la surcharger avec trop de détails (voir Figure 1).

Ensuite, nous avons examiné les différents éléments que nous avions à notre disposition et avons réfléchi à des façons créatives de les utiliser pour créer une map intéressante et stimulante. Nous avons utilisé des textures et des éclairages différents pour donner de la profondeur à la carte, en nous assurant que chaque élément était placé avec soin pour maintenir un certain équilibre (voir Figure 2). Malgré la simplicité de notre map, nous sommes fiers du premier résultat obtenu, tout en sachant qu'il nous reste encore beaucoup de travail pour atteindre notre objectif (voir Figure 3).

2.2 Ennemis

La création d'une intelligence artificielle (IA) sous Unity est un processus complexe qui implique l'utilisation de plusieurs outils et technologies avancées. Parmi ces outils, le Nav Mesh est un élément clé qui permet de générer des chemins de navigation pour les objets dans un environnement 3D. Le Nav Mesh est un maillage de triangles qui représente la géométrie de l'environnement de jeu. En l'utilisant, l'IA peut calculer les meilleurs chemins pour atteindre sa destination tout en évitant les obstacles et les autres objets dans l'environnement. Cela permet à l'IA de se déplacer de manière fluide et réaliste, ce qui améliore l'expérience de jeu pour le joueur.

2.2.1 Les fondamentaux de l'intelligence artificielle

La création d'une IA sous Unity commence par la conception et la modélisation de l'environnement de jeu. Il est important de créer un environnement réaliste avec des obstacles, des chemins et des points d'intérêt pour que l'IA puisse interagir avec son environnement. Ensuite, il faut créer le Nav Mesh en utilisant les outils de navigation de Unity (voir Figure 4).

2.2.2 Pathfinding/Mesh Agent

Tout d'abord, nous avons créé une intelligence artificielle basique qui se déplace là où l'on clique en empruntant le chemin le plus court (voir Figure 5).

2.2.3 Déplacement aléatoire

Ensuite, nous avons amélioré l'IA pour qu'elle puisse se diriger vers le joueur. De plus, nous avons augmenté sa vitesse lorsque le joueur entre dans son champ de détection (indiqué par une sphère jaune) pour donner un effet de poursuite. Lorsque le joueur se trouve dans la sphère rouge, l'IA lance une animation de dialogue. À l'avenir, lorsque nous aurons trouvé l'animation appropriée, l'IA sera en mesure de donner des coups de corps à corps avec la crosse de son arme (voir Figure 6).

2.2.4 Animations

Au niveau des animations, l'IA en possède déjà plusieurs, notamment lorsqu'elle ne bouge pas, nous avons mis une animation qui simule la respiration et les petits gestes que l'on fait lorsqu'on est immobile (voir Figure 7). Il y a aussi une animation de marche et de sprint selon qu'un joueur est dans le rayon de détection ou non. Enfin, lorsque l'IA arrive au niveau d'un joueur, une animation permet à l'IA de simuler une conversation avec le joueur (voir Figure 8). Nous avons mis cette dernière juste afin de voir si cela marchait et nous changerons bien évidemment cette animation. Pour la prochaine soutenance nous devons finaliser les animations de base (marcher, courir, tirer), ainsi qu'implémenter un système de Raycast afin de permettre à l'IA de tirer et de faire des dommages grâce au système de dégâts que nous avons déjà commencé à implémenter.

2.3 Armes

Les armes sont une partie essentielle dans le Gameplay du jeu. C'est pour cela que nous avons implémenté différentes armes. Pour l'instant nous avons créé deux armes : la première est une AK-47 qui est un fusil d'assaut emblématique et mondialement connu dans les FPS, puis notre deuxième arme est une FN-SCAR qui elle, est un fusil d'assaut belge très populaire aussi, grâce à certains jeux comme Fortnite ou bien Call of Duty (voir Figure 9).

Pour ajouter du réalisme au jeu, nous avons dû implémenter des animations aux armes comme par exemple, le joueur peut tenir l'arme dans ses mains ou bien l'animation quand le joueur sort l'arme (voir Figure 10).

2.4 Gameplay

2.4.1 Déplacements

En partant de l'idée d'un jeu type FPS, il fallait donc commencer par implémenter tous les mouvements essentiels tels que le déplacements avant et arrière, le saut et l'accroupissement du joueur.

Pour commencer, afin de rendre un code lisible et compréhensible, nous avons décidé de créer 2 scripts : `PlayerController` et `PlayerMotor`. Le premier script est celui qui détecte les entrées et le second quant à lui va être responsable de la physique du personnage, les déplacements etc... Ainsi ces deux scripts sont complémentaires car le joueur ne peut pas se déplacer sans les entrées et vice versa les entrées sans action n'engendrent rien. Afin de lier les deux scripts nous avons dû utiliser un `[RequireComponent]`

Afin de se déplacer, nous avons utilisé la méthode `GetAxisRaw` pour récupérer la touche du clavier que le joueur a paramétrée afin de se déplacer, puis nous définissons un nouveau `Vector3` qui calcule la vitesse du joueur où nous l'appliquons à une méthode `Move` qui permet au joueur de se déplacer.

2.4.2 Joueur

Pour plus de cohérence avec le style de notre jeu, nous avons dû modifier l'apparence du joueur. En effet, nous voulons faire un jeu style *Call of Duty* ainsi nous avons téléchargé un Asset sur le AssetStore de Unity afin d'avoir les détails du personnage. De plus, nous avons créé un script qui s'appelle `Player` qui va regrouper toutes les informations du joueur. Quand un joueur apparaît sur la map, il a 100 points de vie et tous ses composants sont activés (voir Figure 11). De plus, nous avons implémenté une hitbox pour le joueur. Celle-ci correspond à la zone où les balles ennemies peuvent toucher le joueur et causer des dégâts. Cela permet de rendre le jeu plus réaliste et stratégique, car le joueur doit faire attention à sa position et à ses mouvements pour éviter les tirs ennemis et survivre dans l'environnement hostile du jeu (voir Figure 17).

2.4.3 Respawn

Lorsqu'un joueur arrive sur la map, il apparaît sur le premier point de spawn (point de réapparition), si le joueur meurt il va réapparaître aléatoirement sur un des deux points que nous avons paramétrés au bout de 5 secondes grâce à la méthode `Respawn`, qui se trouve dans le script `Player`. Nous avons défini les points de réapparition afin que lorsque le joueur apparaît il ne tourne pas le dos à l'autre point de spawn. De plus, quand le joueur meurt, tous ses composants sont désactivés, c'est-à-dire qu'il lui est impossible de se déplacer, tirer etc...

2.4.4 Système de tir

Le système de tir était un des points les plus complexes. Premièrement nous avons créé le script `PlayerShoot` qui va nous permettre de définir le système de tir. Dans ce fichier, nous avons implémenté la méthode `Shoot` qui crée un `Raycast` et qui envoie l'information au serveur quand un joueur est touché. Le `Raycast` part de la caméra du joueur vers l'avant et dépend de la portée de l'arme. Nous avons pu rencontrer plusieurs problèmes. Premièrement, quand on tirait sur n'importe quel objet sur la carte, une information était envoyée afin de dire que cet objet était touché. Pour y remédier, nous avons défini un `layer` afin que le `Raycast` ne prenne en compte que le `Layer` d'un joueur et ignore tous les autres. Les `Layers` sont des masques qui forment une famille pour tous les `GameObject` qui les portent. Ensuite, nous avons eu un problème avec les balles, en effet les balles traversaient les murs. Pour corriger ce problème, nous avons modifié notre code pour que le `Raycast` prenne en compte tous les objets mais n'envoie pas d'information au serveur. A noter que la méthode `Shoot` est appelée uniquement du côté client grâce à `[Client]` devant la définition de la méthode. Au contraire, le résultat de cette méthode est `CmdPlayerShot` qui comme son nom l'indique est une commande appelée par le serveur grâce à `[Command]`. Ainsi lorsque le joueur touche un ennemi, celui-ci perd de la vie qui est synchronisée pour l'ensemble des joueurs du serveur.

2.4.5 Interface

Pour l'instant, nous n'avons pas pu consacrer beaucoup de temps sur l'interface utilisateur. Néanmoins, nous avons mis un réticule pour les joueurs qui se connectent à la partie afin qu'ils puissent viser plus facilement. Le seul problème que nous avons rencontré était la présence du réticule dans le menu et en jeu (voir Figure 12).

2.5 Multijoueur

Pour les différentes solutions de multijoueur fonctionnant sur Unity, trois choix se sont offerts à nous : `UNet`, `Photon` et `Mirror`. `UNet` est la solution la plus connue car elle était le système officiel de Unity. Cependant, en 2018, `UNet` a été arrêté la rendant impossible à utiliser. `Mirror` est une API open source, qui permet de mettre en place un système de multijoueur facilement sur Unity.

Elle permet de faire communiquer toutes les instances entre elles. L'utilisation de Mirror nous a paru plus simple, nous l'avons donc privilégiée par rapport à Photon. Sa mise en place sur Unity a été sans difficulté car elle a été directement disponible sur l'asset store.

Ensuite, lorsque la décision de l'API utilisée et sa mise en place sur Unity ont été faites, nous avons pu commencer la mise en place d'un début de multijoueur. La base du fonctionnement de Mirror se fait avec un objet "Network Manager" à placer sur la scène. Cet objet doit contenir un composant du même nom qui permet de gérer grâce à la variable "Player Prefab" l'apparition des joueurs lors de la connexion au serveur. Un autre composant nécessaire est "Network Manager HUD" qui permet d'afficher le HUD de connexion de Mirror (voir Figure 13).

Une fois les composants ajoutés, les points de spawn ont pu être installés. Néanmoins, un premier problème s'est présenté lors de la connexion de 2 joueurs, en effet, une personne pouvait contrôler tous les joueurs. Nous avons donc créé le script PlayerSetup qui vérifie si vous êtes le joueur local, et si ce n'est pas le cas, certains scripts se désactivent. Le second "gros" problème a été au niveau de la synchronisation. Lorsqu'un joueur se déplaçait, les autres ne voyaient aucun changement. C'est grâce au composant NetworkTransform appliqué sur le joueur que la synchronisation est possible. Nous avons donc un multijoueur fonctionnel en local (voir Figure 14).

2.6 Site Web

La finalité pour cette première soutenance était de produire un site fonctionnel, avec plusieurs pages et un style déjà présent et agréable. L'objectif à terme étant d'ajouter à cela des fonctionnalités supplémentaires, telles que la possibilité de se créer un compte et de se connecter, la possibilité de télécharger le jeu ou même des effets visuels mettant en valeur notre site.

Le début du développement de ce site nous a posé un premier problème. En effet, il nous fallait trouver un style qui conviendrait au jeu que l'on proposerait. Nous nous sommes inspirés de plusieurs autres sites de jeux vidéo pour créer un design simple et épuré. Celui-ci se compose d'une barre de navigation horizontale en haut et de pages aérées. Nous avons alors commencé à écrire les fichiers HTML et les scripts CSS correspondants, un travail quelque peu redondant et peu intéressant mais qui s'avérait nécessaire. Le fait de développer les principales pages et de les remplir fut plus rapide que prévu. Nous avons donc décidé de nous avancer sur nos objectifs initiaux et avons commencé à rajouter des fonctionnalités supplémentaires sur le site à l'aide de JavaScript, que

nous avons alors dû apprendre, puis faire des recherches sur les formulaires de connexion et d'inscription pour pouvoir les implémenter des fonctionnalités de base (voir Figure 15).

Nous avons donc trouvé des scripts sur internet et les avons adaptés pour notre site. Nous avons ensuite décidé de rajouter un défilement d'images sur la page d'accueil pour la rendre plus attractive. L'apprentissage du JavaScript et la manipulation de ce langage étaient très intéressants. Adapter les images pour le site fut une tâche plus fastidieuse (voir Figure 16).

3 Planning prévision

Pour la deuxième soutenance, nous envisageons de finir le projet à 70%. Cependant, selon l'avancement de notre projet, nous pourrions nous donner la liberté d'ajouter de nouvelles fonctionnalités. Pour nous ce qui nous semble le plus important c'est d'avoir un vrai jeu fonctionnel avec un multijoueur en ligne et une IA capable de tirer. Évidemment, notre site Web sera plus épuré et apportera de nombreuses informations précieuses tant sur le jeu que sur son développement.

3.1 Prochaine soutenance

Tâches	Soutenance 17-21/04
Interface	60%
Son	70%
Gameplay	80%
Game Design	70%
Animation	80%
Modélisation	60%
IA	70%
Multijoueur	80%
Site Web	90%

4 Conclusion

Pour conclure, nous avons un résultat assez proche des prévisions avec des progrès dans plusieurs domaines.

En effet, le site web a été avancé, nous avons pu aussi assez progresser pour pouvoir rencontrer et évaluer les différents problèmes dans le gameplay. Nous avons présenté un début prometteur de notre intelligence artificielle, qui est capable de se déplacer aléatoirement sur une carte, ainsi que de poursuivre un joueur en utilisant des animations limitées. Le mode multijoueur a également été implémenté avec succès, permettant à deux joueurs de se trouver sur la même carte et de voir les mouvements de l'autre. Cependant, la carte est toujours en cours de construction et des améliorations supplémentaires sont nécessaires pour enrichir les animations disponibles.

5 Webographie

- Mirror : <https://assetstore.unity.com/packages/tools/network/mirror-129321>
- Asset : <https://assetstore.unity.com/packages/3d/environments/low-poly-free-vegetation-kit-176906>

6 Annexes

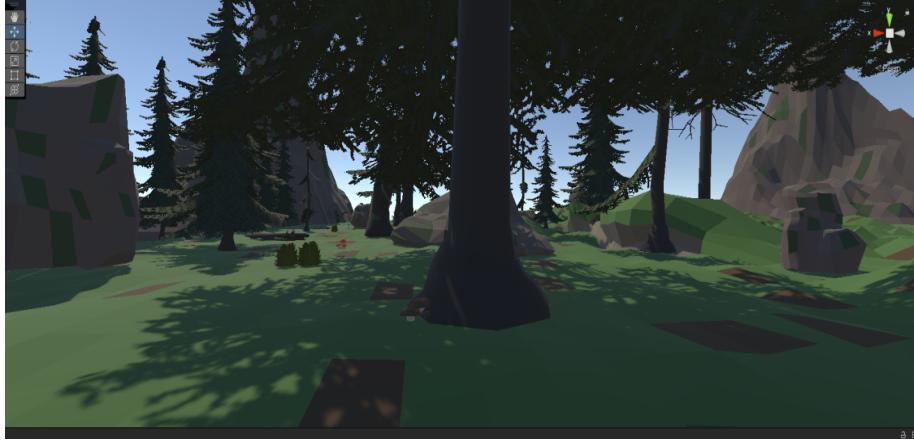


FIGURE 1 – Capture d'écran 1

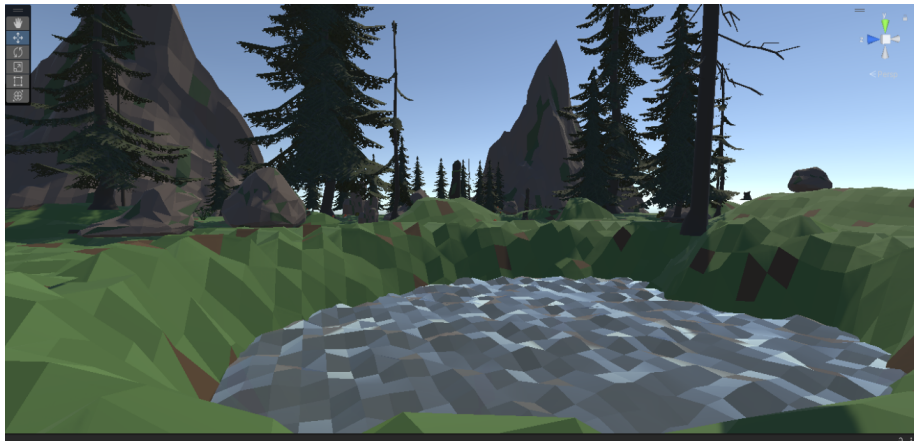


FIGURE 2 – Capture d'écran 2

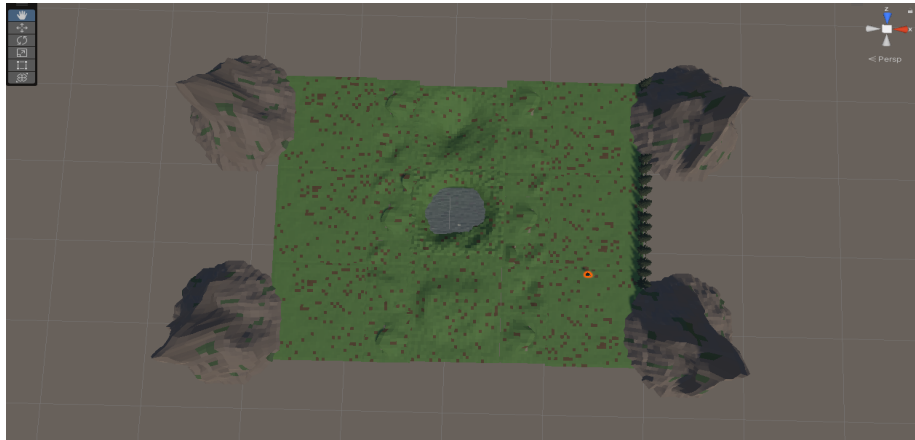


FIGURE 3 – Capture d'écran 3

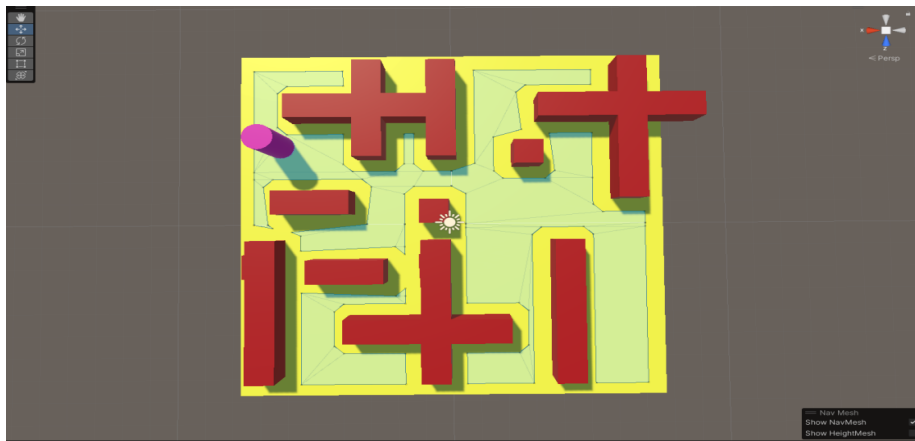


FIGURE 4 – Capture d'écran 4

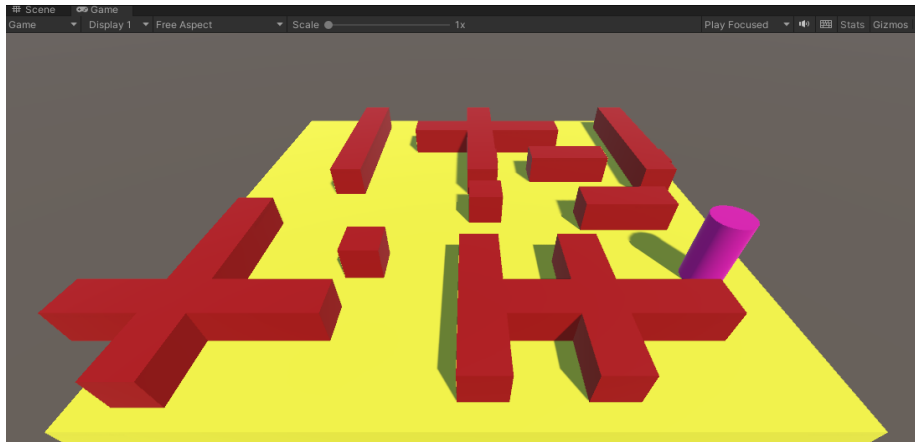


FIGURE 5 – Capture d'écran 5

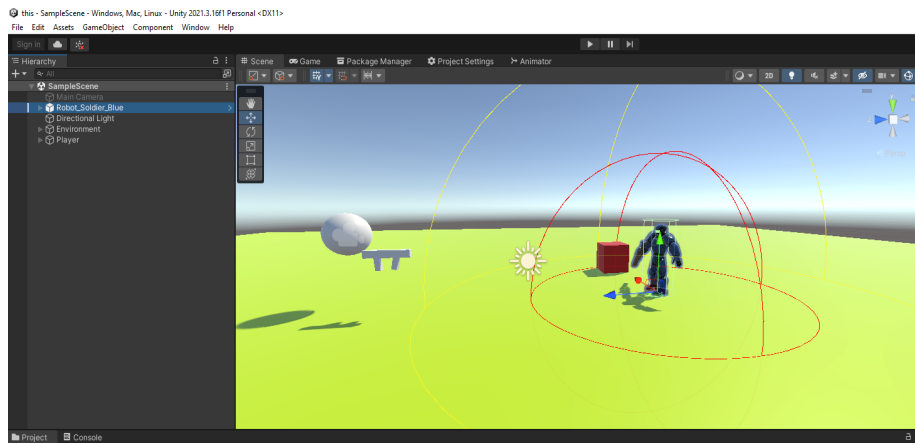


FIGURE 6 – Capture d'écran 6

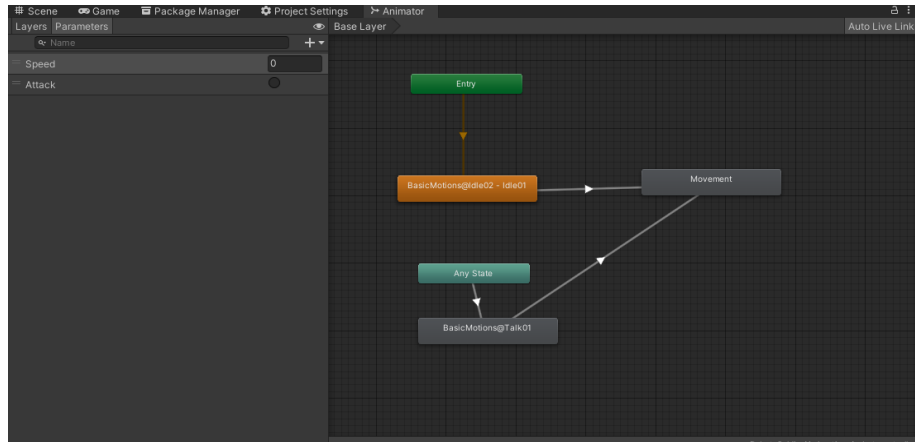


FIGURE 7 – Capture d'écran 7



FIGURE 8 – Capture d'écran 8

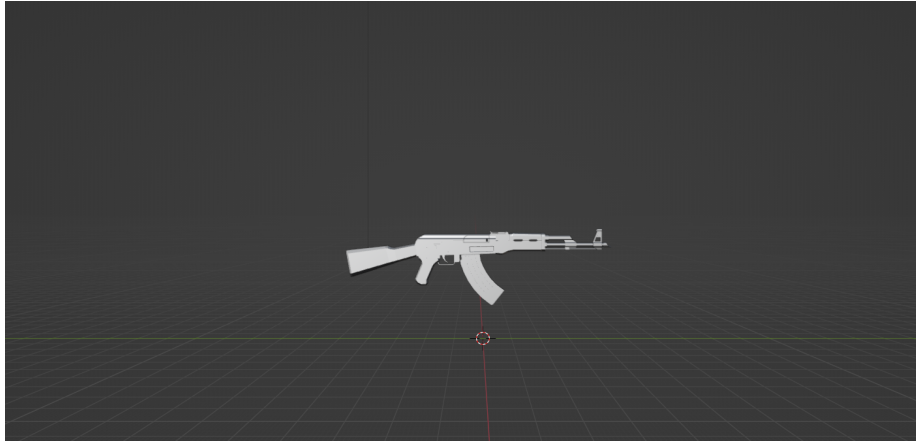


FIGURE 9 – Capture d'écran 9

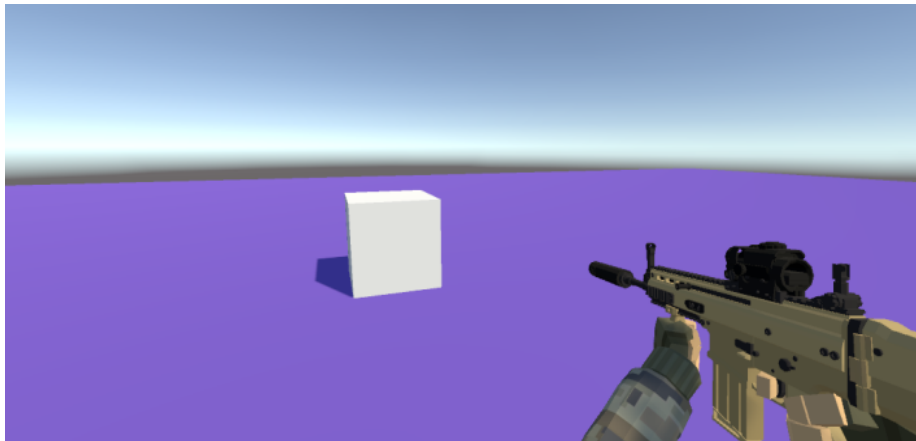


FIGURE 10 – Capture d'écran 10

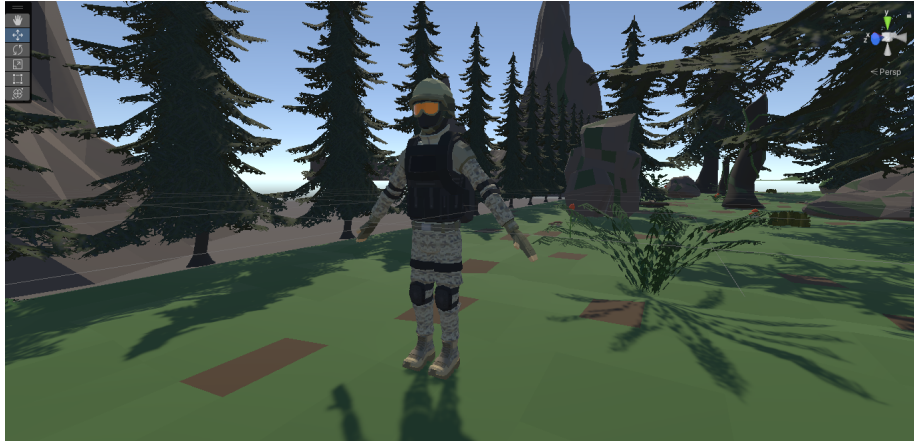


FIGURE 11 – Capture d'écran 11



FIGURE 12 – Capture d'écran 12

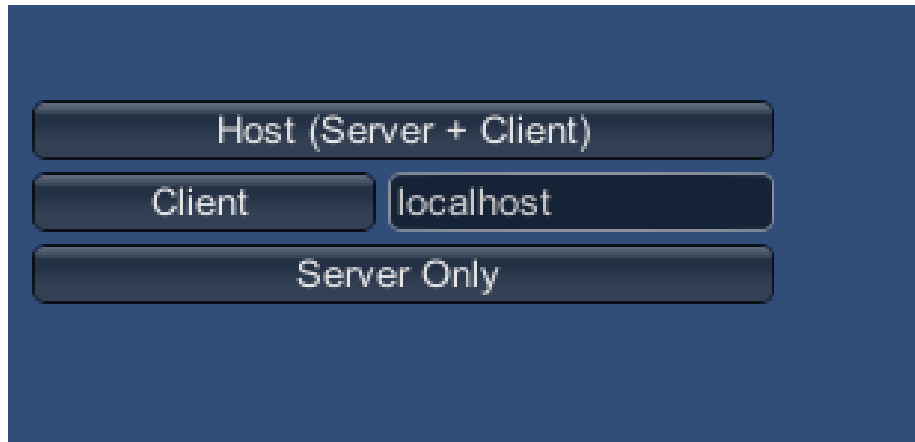


FIGURE 13 – Capture d'écran 13

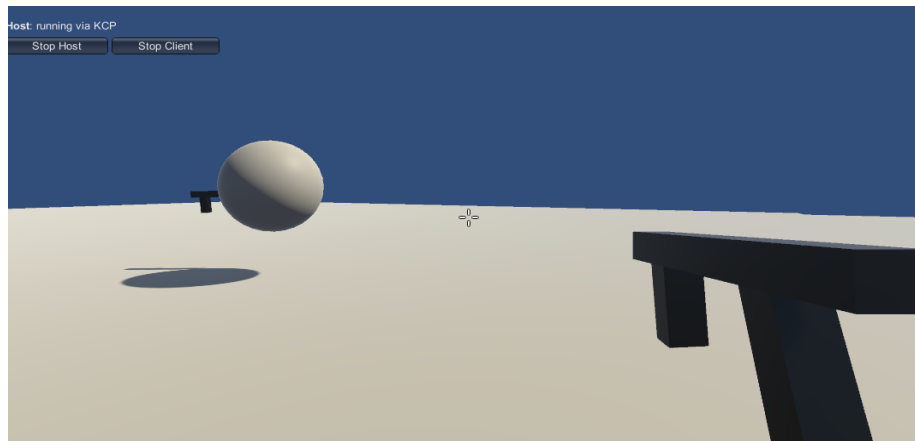


FIGURE 14 – Capture d'écran 14

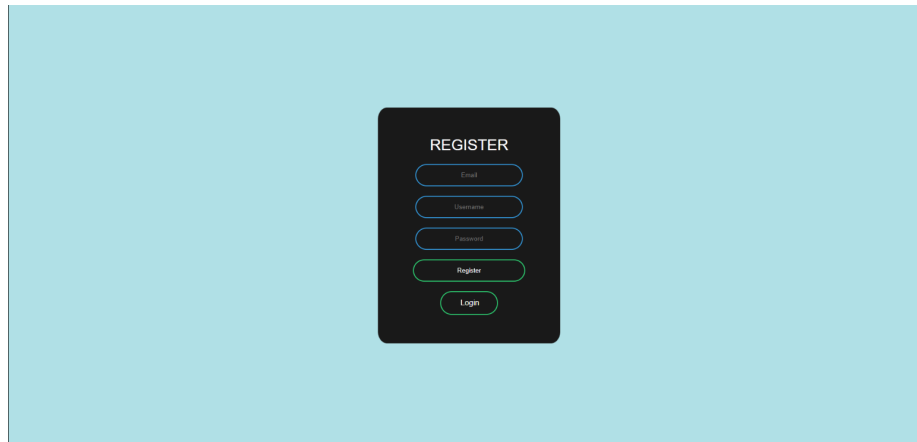
A screenshot of a web application's registration form. The form is a dark gray rectangle centered on a light blue background. It has a title "REGISTER" at the top. Below the title are five input fields: "Email", "Username", "Password", "Register", and "Login". The "Register" and "Login" buttons are green, while the others are dark gray.

FIGURE 15 – Capture d'écran 15



FIGURE 16 – Capture d'écran 16

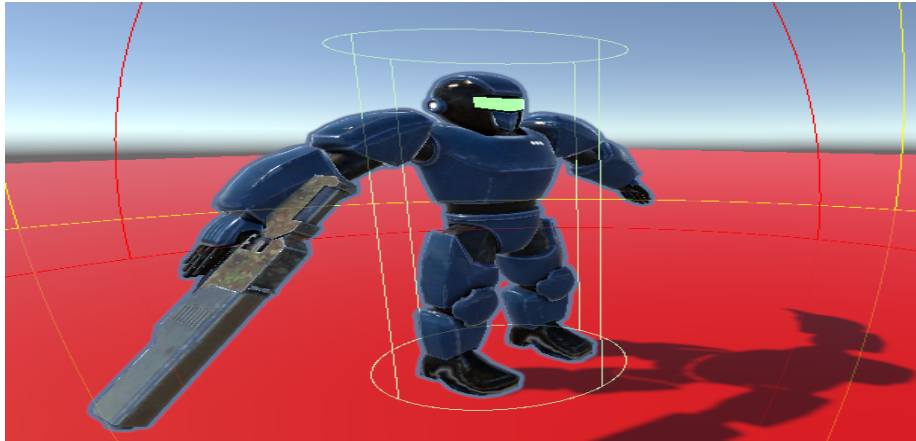


FIGURE 17 – Capture d'écran 17