



VALDORIAM GAMES

RAPPORT DE SOUTENANCE
FINAL



The Throne of Valdoriam

Théodore Pénicaut
Leewen Steinmetz
Giovanni Gast
Romain Bailly
Romario Laurencena

26 mai 2025

Promo 2029

Table des matières

1	Introduction	4
2	Logos	4
2.1	Le logo de Valdoriam Games	4
2.2	Le logo du jeu	5
3	Lore	7
3.1	Cinématiques	7
3.2	Quêtes	7
4	Mécaniques de jeu	8
4.1	Inventaire	8
4.2	Système de détection des objets	8
4.3	Le système d'interface utilisateur	10
4.3.1	La représentation graphique	10
4.3.2	Les actions de l'inventaire	10
5	Conception de niveaux	12
5.1	Recherche bibliographique	12
5.2	Conception de la carte	12
6	Analyse détaillée du système de quêtes	12
6.1	Introduction	12
6.1.1	Début de la quête	13
6.1.2	Progression vers la cité	13
6.1.3	Combat contre le boss et ses sbires	13
6.1.4	Trouver et utiliser un code secret	13
6.1.5	Défendre la ville contre des vagues d'ennemis	13
6.2	Étape 1 : Fuir et fermer la porte	14
6.3	Étape 2 : Traverser la forêt	14
6.4	Étape 3 : Combattre le boss et les ennemis	14
6.5	Étape 4 : Trouver le code et fermer la porte	14
6.6	Étape 5 : Défendre la ville contre les vagues d'ennemis	15
6.7	Résumé final	15
6.7.1	Représentation de la carte	17
6.8	Minimap	18
7	Répartition des tâches	20
8	Avancement	21
9	Mouvements, Caméra et Animations	22
9.1	Mouvements	22
9.2	Caméra	22
9.3	Les Animations	23

10 Les Classes	25
10.1 Les Classes C# Utilitaires	25
10.2 Les Classes Jouables	25
11 Les VFX et particules	27
11.1 Les particles systems	27
11.2 Les VFX Graphs	28
12 Système d'attaque	29
12.1 Objectifs	29
12.2 Conception	30
12.2.1 Équiper/déséquiper une arme	30
12.2.2 Gestion des attaques	30
13 Interface utilisateur	31
13.1 Menu Principal	31
13.2 Menu Pause	33
13.3 Choix des classes jouables	35
13.4 Difficultés rencontrées	36
14 Site Internet	37
14.1 Page d'accueil	37
14.2 Page "A propos de Valdoriam Games"	38
14.3 Page "A propos des créateurs"	38
14.4 Page "Nous contacter"	39
14.5 Page d'annexe	39
15 Multijoueur	40
15.1 Implémentation	40
15.2 Création du terrain et synchronisation	40
15.3 Mode de jeu Multijoueur	41
15.4 Difficultés	42
16 IA	44
16.1 IA ennemis	44
16.1.1 États comportementaux	44
16.1.2 Animation des IA ennemis	45
16.1.3 Détection visuelle	45
16.2 IA alliées	45
16.2.1 Structure des données	45
16.2.2 Gestion centrale : <code>Manger_Commandement</code>	45
16.2.3 Ordres disponibles	46
16.2.4 Comportement et animation : <code>IA_Allie_Comportement</code>	46
16.2.5 Interface de commande : <code>Comm_Slot</code>	46
16.2.6 Interaction et Layers	46
16.2.7 Détection spatiale	46
16.3 Analyse du système existant	47
16.4 Limitations observées	47
16.5 Axes d'amélioration	47

16.5.1 Architecture logicielle	47
16.5.2 Gameplay et contrôle	47
16.5.3 IA et navigation	47
16.5.4 Optimisation	48
16.5.5 Expérience utilisateur (UX)	48
16.6 Perspectives d'évolution	48
16.7 Résumé IA	48
17 Conclusion	49

1 Introduction

Fondé par une équipe passionnée de développeurs, Valdoriam Games est un studio indépendant dédié à la création d'expériences immersives mêlant narration riche et gameplay exigeant. Leur premier projet, **The Throne of Valdoriam**, plonge les joueurs dans un monde fantastique sombre et envoûtant, inspiré des plus grands Souls-like tout en apportant une identité unique.

Dans ce royaume déchiré par une guerre ancestrale, les joueurs incarnent un héros maudit, en quête du trône légendaire de Valdoriam. Entre combats tactiques éprouvants, énigmes mystérieuses et paysages à couper le souffle, le jeu allie difficulté stratégique et exploration approfondie. L'univers, nourri par un lore dense et une direction artistique gothique, invite à découvrir les secrets d'une civilisation oubliée, où chaque épée rouillée et chaque fresque effacée racontent une histoire.

Avec **The Throne of Valdoriam**, Valdoriam Games ambitionne de marquer les esprits des amateurs du genre, en proposant une aventure aussi impitoyable que mémorable, où la persévérance et la curiosité sont les clés d'une victoire éphémère... mais glorieuse.

2 Logos

2.1 Le logo de Valdoriam Games



FIGURE 1 – Logo officiel de Valdoriam Games.

Le logo de Valdoriam Games incarne l'essence de notre studio de jeux vidéo : son design épuré, dominé par la silhouette d'une forteresse stylisée, évoque à la fois la solidité,

l'exploration et l'univers médiéval-fantastique qui inspire notre création.

Ce choix minimaliste assure une lisibilité optimale, une identité forte et mémorable, ainsi qu'une adaptabilité sur tous les supports. En associant simplicité et impact visuel.

Notre logo en noir et blanc n'est pas un hasard, mais un choix réfléchi. Cela permet d'avoir un logo intemporel et supprimant toutes distractions des couleurs, mettant en avant l'essence même du logo et renforçant son identité.

En tant que petite entreprise, nous avons fait le choix d'intégrer directement le nom « Valdoriam Games » dans notre logo. Cela nous permet d'accroître notre visibilité et notre reconnaissance auprès du public. Contrairement aux grandes marques pouvant se contenter des symboles seuls, cela permet que chaque apparition de notre logo soit associée à l'image de notre entreprise.

2.2 Le logo du jeu



FIGURE 2 – Logo officiel de The Throne of Valdoriam.

Ce logo incarne parfaitement l'essence de notre jeu, The Throne of Valdoriam, en adoptant un style épuré et symbolique. Son design minimaliste mais puissant permet une lecture immédiate et une forte identité visuelle, tout en capturant les thèmes majeurs de l'univers du jeu.

Le trône, élément central du visuel, fait directement écho au titre du jeu. Il incarne le pouvoir absolu, la domination et l'ambition qui animent le protagoniste, représentant l'objectif ultime de son ascension.

L'épée, placée au centre du trône, est un symbole fort qui traduit la lutte acharnée et les épreuves que le protagoniste devra affronter tout au long de son périple. Elle illustre les combats auquel il sera confronté tout au long de l'aventure .

Les couleurs sombres, dominées par le rouge et le noir, renforcent cet univers impitoyable. Le rouge, symbole de sang et de passion, évoque la violence des batailles et le prix à payer pour le pouvoir. Le noir, quant à lui, incarne, la mort et la menace omniprésente qui plane sur le protagoniste.

Ensemble, ces deux éléments matérialisent la dure et impitoyable quête de pouvoir du personnage principal, où chaque erreur pourrait lui être fatal.

3 Lore

Pour le scénario, nous avons décidé de finaliser à 100 % le scénario du jeu. Pour cela, nous avons rédigé le document suivant qui organise les cinématiques et les différentes quêtes. Ce document nous a servi de feuille de route lors de la conception du système de quêtes.

3.1 Cinématiques

- **Cinématique 1** : Présentation de l'histoire du monde.
- **Cinématique 2** : Introduction de la situation initiale : le personnage principal vit en paix dans un orphelinat avec ses amis dans la cité d'Isilidur.
- **Cinématique 3** : Le comte Vandover ordonne un raid sur l'orphelinat où vit Nilhum.
- **Cinématique 4** : Nilhum se réfugie dans une auberge après son combat et rejoint, par vengeance, une guilde.
- **Cinématique 5** : Nilhum devient un membre officiel de la Guilde et peut enfin commencer sa vengeance après deux ans.
- **Cinématique 6** : Révélation des véritables origines de Nilhum et des raisons de l'attaque de l'orphelinat par le comte Vandover.
- **Cinématique 7** : Après ses actes de bravoure, Nilhum reçoit une mission de sa famille : récupérer le trône de Valdoriam.

3.2 Quêtes

- **Quête 1 : Tutoriel.** Le joueur ont différentes compétences selon la classe choisie :
 - **Chevalier** : Attaquer sauter et taillade circulaire
 - **Mage** : Lancer des sorts (protection et attaque).
 - **Assassin** : invisibilité et écran de fumée.
- **Quête 2 :** Missions à réaliser :
 - course poursuite
 - Recherche d'un code
 - Combat contre une Sentinel (boss)
Tous les joueurs se familiarisent également avec les déplacements via une course-poursuite avec des squelettes.
- **Quête 6 : Protection.** Défense de la cité d'Islim contre des vagues d'ennemis.
 - Défaite : le jeu relance une nouvelle partie

4 Mécaniques de jeu

4.1 Inventaire

On peut diviser le développement de l'inventaire en trois étapes :

- Le système de détection des objets
- Le système d'interface utilisateur
- La représentation graphique
- Les actions de l'inventaire

Mais avant d'expliquer comment détecter des objets dans une scène, abordons une notion capitale pour l'inventaire, qui en constitue la base.

Cette notion est celle des **ScriptableObjects**. Un ScriptableObject dans Unity est un type spécial d'objet qui permet de stocker des données de manière centralisée et réutilisable, sans dépendre d'une scène ou d'un objet de jeu. Il est utilisé pour gérer des configurations, des paramètres de jeu ou des bases de données d'objets.

Dans notre jeu, pour représenter les objets de l'inventaire, nous avons utilisé ce type spécial d'objet. En pratique, on peut assigner aux ScriptableObjects des variables, qui sont l'équivalent de propriétés pour les classes.

Dans notre ScriptableObject, qui représente un objet, nous avons assigné cinq variables : le nom, une description, un visuel, un modèle et un type.

Maintenant que nous avons posé les bases, nous allons pouvoir passer au système de détection des objets.

4.2 Système de détection des objets

Pour détecter des objets, nous avons utilisé dans notre jeu ce qu'on appelle un **SphereCast**. Un SphereCast dans Unity est une méthode qui permet de projeter une sphère virtuelle depuis un point d'origine dans une direction donnée, sur une certaine distance. Pendant son déplacement, la sphère détecte les objets qu'elle traverse ou touche, renvoyant des informations sur ces collisions, comme l'objet touché et la position d'impact.

Maintenant que nous savons comment détecter les objets, il faut être capable de détecter uniquement ceux qui nous intéressent.

Ici, nous voulons détecter les objets que nous avons définis au préalable comme des items. Pour cela, nous avons utilisé des **couches**.

Une couche est une catégorie attribuable aux objets de jeu pour gérer leur interaction avec certains systèmes comme le rendu, la physique et les scripts. Elle permet de simplifier la détection dans les scripts en appliquant des opérations comme des **Raycasts** sur des couches spécifiques.

Unity prend en charge jusqu'à 32 couches, dont certaines sont réservées par défaut, comme "Default" et "UI". Ici, nous avons donc créé une couche "Objet" et l'avons attribuée aux objets voulus.

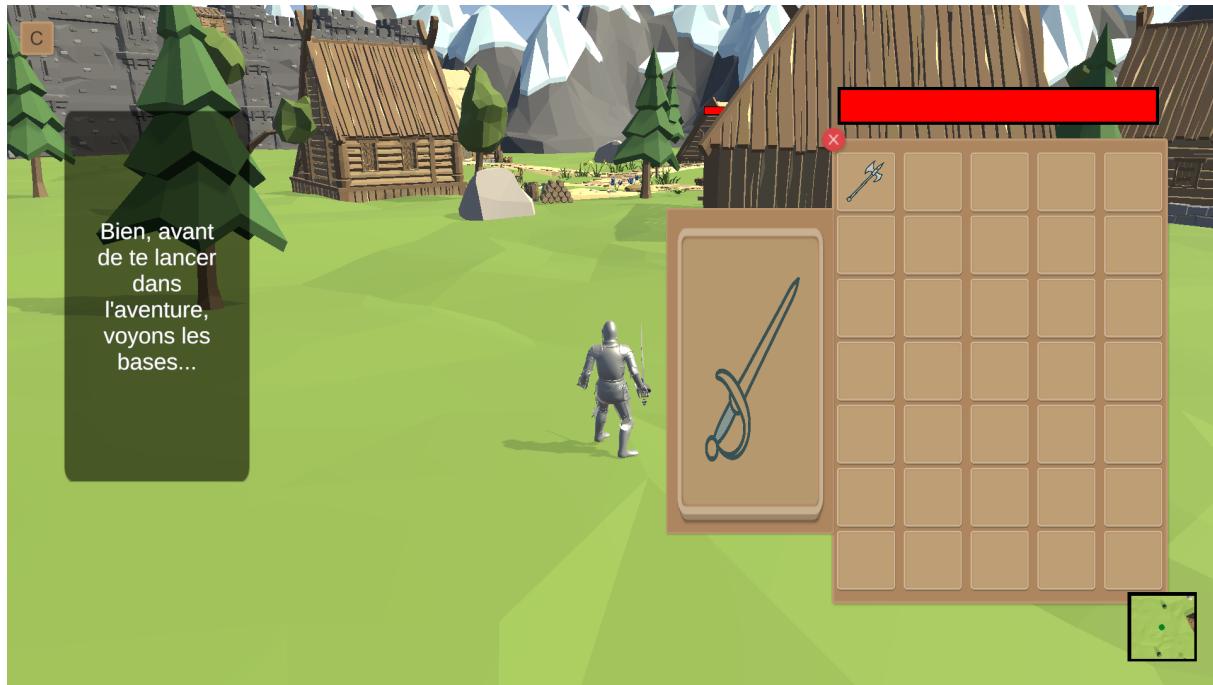


Exemple de détection d'objet^{*}

Pour finaliser le système de détection des objets, il faut décider comment représenter informatiquement l'inventaire.

Tout d'abord, rappelons que nous avons décidé de représenter un objet par un `ScriptableObject`, que nous avons appelé "`Objet_Script_RL`". Pour représenter l'inventaire, nous allons attacher à un objet de jeu vide un script qui contient une liste d'objets "`Objet_Script_RL`". Chaque objet aura donc la couche "`Objet0`" et un script contenant le `ScriptableObject` correspondant.

En résumé, le **SphereCast** détecte l'objet si celui-ci a la bonne couche. Suite à une action X du joueur, l'objet est ajouté dans la liste qui représente l'inventaire. Lorsqu'on fait cela, l'objet est évidemment détruit.



Un exemple d'arme équipée

4.3 Le système d'interface utilisateur

4.3.1 La représentation graphique

Pour la représentation graphique, nous avons utilisé une image que nous avons agrandie. Dans cette image, nous avons placé 32 cases qui ont pour source une image carrée beige. Chacune de ces cases est un bouton, c'est-à-dire qu'on peut lui associer des méthodes lorsqu'on clique dessus, en précisant simplement de quel script proviennent ces méthodes.

Pour faciliter la création de la partie graphique de l'inventaire et pour aligner parfaitement chaque case, nous avons utilisé un composant de l'interface utilisateur Unity : **Vertical Layout Group**. Ce composant permet d'aligner chaque case et de définir l'espace entre elles. Il permet de gagner du temps et d'obtenir un rendu graphique plus soigné.

Maintenant que nous avons parlé de la création de l'aspect graphique de l'inventaire, nous allons pouvoir aborder comment lier la liste qui représente l'inventaire et ces **Canvases**, qui forment l'aspect visuel de l'inventaire.

Pour cela, il suffit de parcourir la liste (que nous allons appeler "contenu") et d'effectuer un **GetChild(i)** pour attribuer à chaque case le visuel de l'objet correspondant.

4.3.2 Les actions de l'inventaire

Avant d'aborder l'implémentation des actions pour chaque objet, il est important de rappeler les différents types d'objets et les actions possibles.

Les types d'objets :

- Armes

Les actions possibles :

- Poser
- Détruire
- Équiper

Évidemment, quel que soit son type, chaque objet peut être posé ou détruit.

Maintenant, passons à l'implémentation de ces actions. Nous ne détaillerons pas l'aspect graphique des actions, car il est très similaire à celui de l'inventaire.

L'action "Détruire" Il suffit simplement de détruire l'objet avec l'instruction **Destroy()** et de rendre son visuel transparent.

L'action "Poser" On définit un objet de jeu vide, que nous allons appeler "point_de_dépose", puis on prend l'objet et on le fait apparaître dans la scène en utilisant l'instruction **Instantiate()** sur la variable modèle de l'objet.

Il est important de noter que le modèle doit lui-même contenir un script avec le ScriptableObject et la bonne couche. Il faut donc créer un modèle préalablement configuré.

Le joueur devra choisir sa classe parmi les différentes options proposées, chacune offrant des capacités uniques adaptées à son style de jeu.



Exemple d'actions disponibles

Système de commandement de troupes

Nous avons décidé d'implémenter un système de commandement de troupes. Pour cela, un menu dédié a été conçu, ce menu permet au joueur de gérer ses troupes de manière intuitive, en offrant plusieurs fonctionnalités essentielles : le recrutement de nouvelles unités, ainsi que la possibilité de renvoyer celles qui ne sont plus nécessaires.

Une fois recrutées, ces troupes ne se contentent pas de suivre passivement le joueur. Elles peuvent être affectées à différents modes de comportement, en fonction de la situation ou de la stratégie choisie. Nous avons mis en place trois modes principaux :

- **Suivre** : les troupes restent proches du joueur et le suivent automatiquement dans ses déplacements.
- **Attaquer** : elles engagent les ennemis de manière autonome dès qu'un combat est détecté à proximité.
- **Défendre** : elles se positionnent autour d'une zone ou d'un objectif spécifique et protègent activement cet emplacement contre toute menace.

Ce système permet au joueur de personnaliser sa façon de jouer, en adoptant un style plus offensif, défensif ou tactique selon les besoins.

Mode Solo

Dans le mode solo, le joueur incarne le personnage principal et suit sa quête pour retrouver son trône. Cette aventure se déroule dans une carte de type open world ou semi-monde ouvert, où il découvrira de nouveaux territoires à explorer. Le joueur sera confronté à une plusieurs variété d'ennemis, allant des simples squelettes aux puissants boss qui règnent sur ces royaumes.

5 Conception de niveaux

5.1 Recherche bibliographique

Pour concevoir la carte du monde de *The Throne of Valdoriam*, nous avons utilisé des assets disponibles sur l'Asset Store de Unity, notamment ceux de **Synthy Studios**, qui fournissent les éléments nécessaires comme des murs, du gravier et des maisons.

5.2 Conception de la carte

Initialement, nous avions prévu d'implémenter un système de souterrains sous le château, afin d'enrichir l'exploration et d'introduire des mécaniques de quête ou de cachettes secrètes. Cependant, pour des raisons de complexité de développement, nous avons finalement décidé de ne pas intégrer cette partie dans le jeu. Cette décision nous a permis de concentrer nos efforts sur des zones déjà existantes.

6 Analyse détaillée du système de quêtes

6.1 Introduction

Le système de quêtes que vous avez conçu propose une progression narrative claire et immersive pour le joueur, décomposée en plusieurs étapes successives, chacune avec ses propres objectifs et mécaniques de jeu. Ce document présente une analyse complète

du déroulement global des quêtes, puis une étude détaillée de chaque étape dans son fonctionnement et sa conception.

6.1.1 Début de la quête

Le joueur commence dans une situation tendue où il doit fuir un danger immédiat. L'urgence est palpable : le joueur doit fermer une porte derrière lui pour ralentir ou bloquer une menace, symbolisée par des vagues d'ennemis. Ce premier objectif donne le ton : il ne s'agit pas simplement d'une promenade, mais d'une fuite avec un fort enjeu.

6.1.2 Progression vers la cité

Une fois cette première étape accomplie, la quête guide le joueur vers un nouveau défi : traverser une forêt. La forêt, souvent dans les jeux d'aventure, est un espace à la fois mystique et dangereux, et ici elle sert de transition vers une zone plus sécurisée, la cité. Le passage d'une porte matérialise cette étape, et le joueur doit physiquement franchir ce seuil pour progresser.

6.1.3 Combat contre le boss et ses sbires

À l'arrivée dans la cité, l'intensité augmente encore : le joueur est confronté à un boss puissant et ses sbires. Ce boss représente un obstacle majeur qui empêche la progression du joueur et menace la sécurité de la cité.

- Le boss se déplace, poursuit le joueur, attaque à portée,
- Il invoque périodiquement des sbires pour complexifier le combat,
- Le joueur doit réussir à vaincre ce boss pour pouvoir continuer.

Ce combat agit comme un véritable point culminant de la quête, avec un enjeu clair : la survie de la cité.

6.1.4 Trouver et utiliser un code secret

Après la victoire contre le boss, le joueur doit accomplir une tâche plus stratégique : retrouver un code dans le château et utiliser ce code pour fermer un second rempart, une autre porte qui protège la cité.

Ce passage introduit une dimension d'exploration et de réflexion. Ce n'est plus qu'un simple combat, mais aussi une énigme, un mini-puzzle à résoudre. Trouver ce code est crucial pour empêcher les ennemis de pénétrer davantage.

6.1.5 Défendre la ville contre des vagues d'ennemis

Enfin, une fois la porte fermée, un portail apparaît : il s'agit d'un nouveau défi, qui consiste à défendre la ville contre plusieurs vagues d'ennemis successives. Cela transforme la quête en une sorte de mode « survie » ou « défense de base », où le joueur doit affronter des assauts répétés, tester ses compétences de combat et sa stratégie.

6.2 Étape 1 : Fuir et fermer la porte

La quête débute avec un message simple mais puissant : « Fuis et ferme la porte ». C'est une consigne directe qui fait appel au sens d'urgence.

- Le panel UI est activé pour afficher ce message.
- Les ennemis (vagues) sont activés, signalant la présence d'une menace immédiate.
- Le joueur doit non seulement fuir, mais aussi veiller à ce que la porte soit bien fermée derrière lui.
- La progression vers l'étape suivante ne se fait que si deux conditions sont réunies :
 - Le joueur a franchi la porte (`Passer_porte`),
 - La porte est bien baissée (fermée) (`porte_baisser0`).

Cette étape est conçue pour forcer le joueur à interagir avec l'environnement de manière précise : ce n'est pas juste un point de passage, mais un acte stratégique qui conditionne la survie.

6.3 Étape 2 : Traverser la forêt

Le message devient plus narratif : « Traverse la forêt pour atteindre la cité ».

- Le joueur est guidé vers un nouvel objectif spatial et narratif.
- Le script attend que le joueur franchisse une nouvelle porte (`porte_passer2`), qui symbolise la sortie de la forêt et l'entrée dans la cité.
- Ce passage est plus une transition narrative et physique, qui pose le décor et prépare le joueur au combat à venir.
- Le texte est clair et simple, mais crée une attente d'un défi plus grand.

6.4 Étape 3 : Combattre le boss et les ennemis

C'est ici que la tension monte réellement. La cité est sous attaque, et le premier rempart a cédé.

- Le script active l'ennemi principal et son boss,
- Un message très immersif est affiché : « Le premier rempart a cédé sous l'armée des morts. Trouvez le code dans le château pour fermer le second rempart. » Ce message est à la fois une mise en contexte et une incitation à agir.
- Le joueur doit combattre le boss, qui invoque des sbires et attaque.
- La quête attend la mort du boss (`est_mort`), avant de progresser.
- Le panel de texte s'adapte en taille pour afficher plus d'information, montrant qu'ici la narration est plus riche et détaillée.

Ce moment est crucial : c'est un combat décisif, la bataille clé qui influence la suite du jeu.

6.5 Étape 4 : Trouver le code et fermer la porte

Le boss est vaincu, mais la cité n'est pas encore sauvée.

- Le joueur reçoit une nouvelle consigne : « Pour sauver la ville, notez le code dans le château et utilisez-le pour fermer la porte du second rempart ».
- Cette étape introduit un nouvel objectif, plus intellectuel : trouver un code (probablement dans un puzzle ou une zone spécifique).
- Le joueur doit ensuite utiliser ce code pour fermer la porte (`porte_baisser1`).
- Le système attend que cette porte soit bien baissée avant de passer à la suite.

- C'est une étape clé dans la narration qui mêle exploration, résolution d'énigmes, et interaction avec l'environnement.
- Le boss est détruit du jeu (via `Destroy(Boss)`), montrant que cette phase de combat est terminée.

6.6 Étape 5 : Défendre la ville contre les vagues d'ennemis

Dernière étape visible dans ce script :

- Un portail apparaît dans la cité, marquant un nouveau défi.
- Le joueur doit défendre la ville contre des vagues successives d'ennemis.
- Le message affiché est clair : « Un portail est apparu, défendez la ville contre les vagues d'ennemis ».
- La quête principale est marquée comme terminée (`activation_Quets = false`), mais une nouvelle phase de gameplay commence (`quets3 = true`), qui semble être gérée par un autre script (probablement `Quets3`).
- Cette phase ajoute un aspect défensif et stratégique, souvent très apprécié dans les jeux, qui teste la résistance du joueur dans un contexte plus dynamique.

6.7 Résumé final

Votre système de quêtes est conçu comme une progression narrative et ludique cohérente, avec :

- Un début intense, marqué par une fuite sous pression,
- Une transition immersive vers un nouveau lieu,
- Un combat épique contre un boss et ses sbires,
- Une phase d'exploration et résolution d'énigmes via la recherche et l'utilisation d'un code,
- Et enfin une phase de défense où le joueur doit tenir contre des vagues d'ennemis.

Chaque étape est soigneusement pensée pour engager le joueur dans un défi différent : action rapide, exploration, combat tactique, puzzle, puis survie.

Le système utilise efficacement les interfaces utilisateur pour communiquer les objectifs, ajuste dynamiquement les éléments de jeu (activation/désactivation d'ennemis, boss, portes), et repose sur des variables booléennes pour gérer la progression.

1. **Les quartiers résidentiels.**
2. **Le château avec son mur d'enceinte.**
3. **Les souterrains du château.**

Voici la carte qui devait représenter l'ensemble des zones explorables dans le jeu :



À l'origine, notre intention était de proposer un vaste monde ouvert pour renforcer l'immersion et laisser une grande liberté d'exploration au joueur. La carte comportait différents environnements : forêts, plaines, montagnes, rivières, et plusieurs petits villages . Chaque zone devait comporter des points d'intérêt uniques, comme des quêtes secondaires, des personnages non-joueurs (PNJ), des objets à collecter ou encore des événements spécifiques.

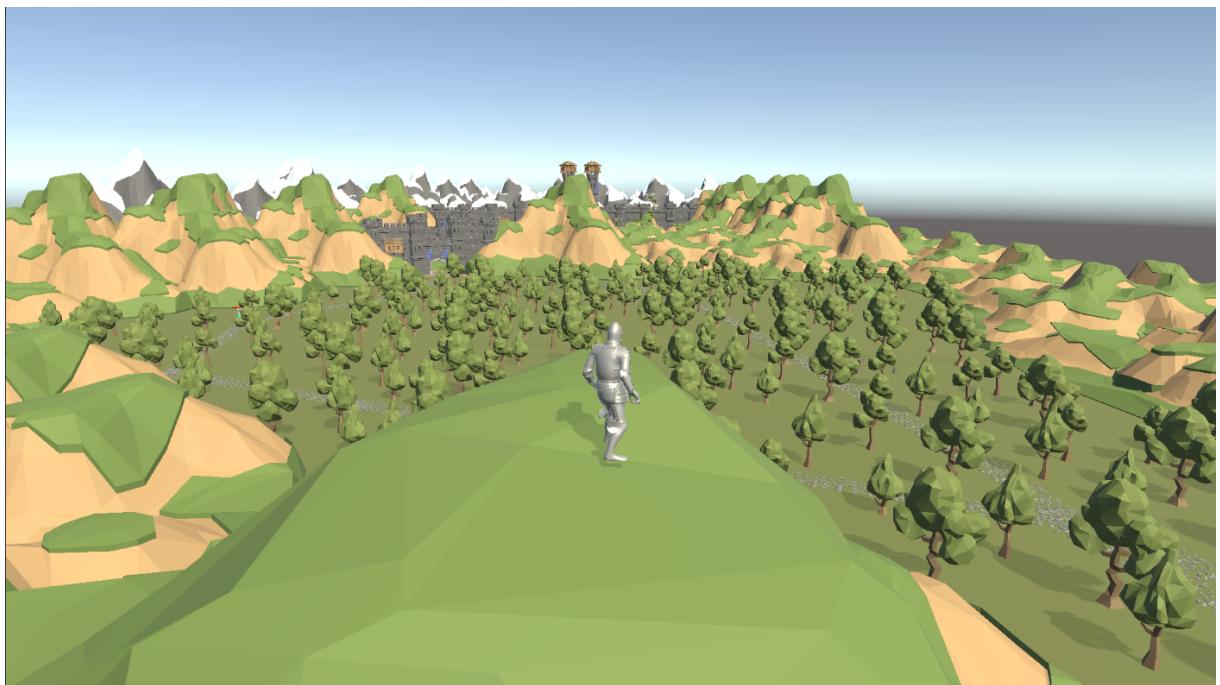
Cependant, au cours du développement, nous avons constaté que la taille de la carte posait plusieurs problèmes. D'une part, sa grande étendue nécessitait une charge de travail importante pour créer du contenu riche et cohérent dans chaque zone. D'autre part, certaines parties restaient trop vides, ce qui diminuait le dynamisme et l'expérience de jeu en donnant une impression d'un monde agrandis pour pas grand choses, sans densité ni interactions significatives.

Nous avons donc pris la décision de réduire considérablement la zone jouable. Cette réduction nous a permis de concentrer nos efforts sur une zone plus restreinte mais plus dense et vivante, en assurant une meilleure qualité dans le level design, les quêtes, et l'environnement sonore et visuel.

6.7.1 Représentation de la carte



Vue Panoramique de la zone de tutoriel



Vue Panoramique de la Grande Fôret



Vue Panoramique de la muraille



Vue Panoramique de la ville

6.8 Minimap

Nous avons par ailleurs pris la décision de ne pas implémenter une carte complète et visible de l'ensemble du monde de jeu. Initialement, nous avions envisagé d'intégrer une carte globale accessible par le joueur à tout moment, lui permettant de visualiser les différentes zones explorées, les objectifs actifs, ainsi que les points d'intérêt importants (villages, quêtes, ressources, etc.).

Mais nous avons finalement opté pour une solution plus légère et mieux intégrée à notre approche de gameplay : l'ajout d'une minimap placée dans l'angle en bas à droite de l'écran.



Cette minimap se concentre uniquement sur l'environnement immédiat du joueur, affichant les éléments essentiels comme la position actuelle, les obstacles majeurs, ou encore certains repères proches.

Le choix d'implémenter seulement une minimap offre une information rapide et directe sans interrompre le rythme de jeu. Ainsi, la minimap remplit son rôle de soutien à la navigation sans nuire à l'aspect immersif du jeu. Cette décision s'inscrit dans notre volonté d'offrir une interface épurée et fonctionnelle, au service de l'expérience du joueur.

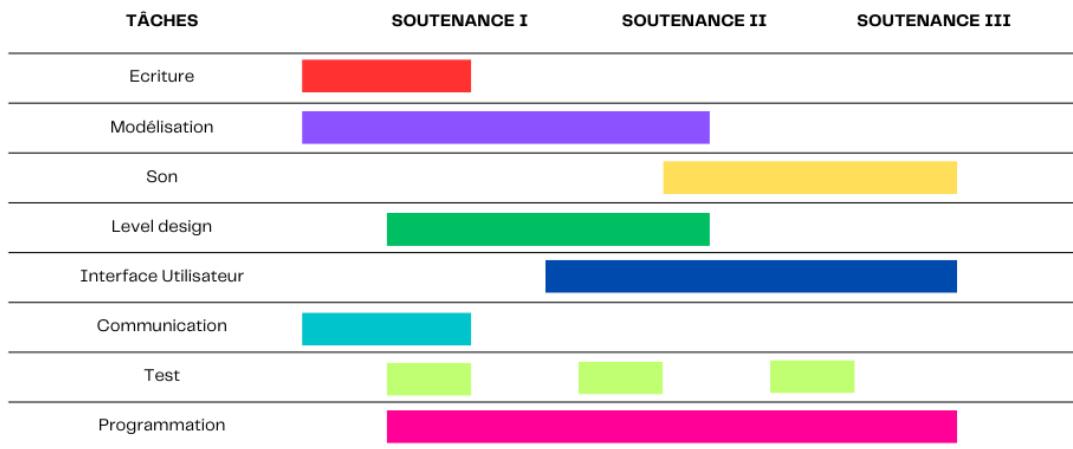
7 Répartition des tâches

Répartition des tâches : deux personnes par tâche: (R)esponsable & (S)uppléant					
Tâches	Login1	Login2	Login3	Login4	Login5
Projet	theodore.penicaut	leewen.steinmetz	romario.laurencena	romain.bailly	giovanni.gast
Edition de niveaux				R	S
Ecriture		S	R		
Modélisation	R			S	
Communication		R	S		
Test	S				R
Son			R		S
Interface utilisateur	R	S			
IA		S			R
Réseau			S	R	

Nous avons globalement respecté la répartition des tâches que nous nous étions fixées (tableau ci-dessus). Cependant, les rôles d'édition des niveaux et de modélisation ont été principalement pris en charge par Romario.

8 Avancement

THE THRONE OF VALDORIAM



9 Mouvements, Caméra et Animations

9.1 Mouvements

Pour implémenter les déplacements, Nous avons regardé plusieurs tutoriels et testé différentes approches. Nous en avons essayé deux avant de nous décider. Ce système est basé sur l'orientation de la caméra, ce qui signifie que le mouvement du joueur est toujours dirigé par la caméra. Peu importe où elle est tournée, les commandes de déplacement (avant, arrière, gauche, droite) seront toujours appliquées correctement en fonction de la direction dans laquelle la caméra regarde. C'est une méthode qui nous permet d'avoir un contrôle intuitif du joueur, nécessaire dans ce type de jeu où la réactivité est essentielle.

9.2 Caméra

pour notre jeu, nous avons opté pour une caméra à la troisième personne, centrée sur le joueur. Ce type de caméra est très répandu dans les jeux de type Souls-like, car il permet une immersion optimale et une grande maniabilité. L'orientation de la caméra joue un rôle clé, car elle définit les axes de mouvement du joueur. Sur le plan technique, Nous avons utilisé Cinemachine, un package performant pour la gestion des mouvements de caméra. Cet outil offre de nombreuses fonctionnalités, telles que : — La création d'une caméra qui suit dynamiquement un GameObject. — La possibilité de modifier facilement la distance entre la caméra et le joueur. — L'ajustement de la vitesse à laquelle la caméra se déplace en réponse aux mouvements de la souris.

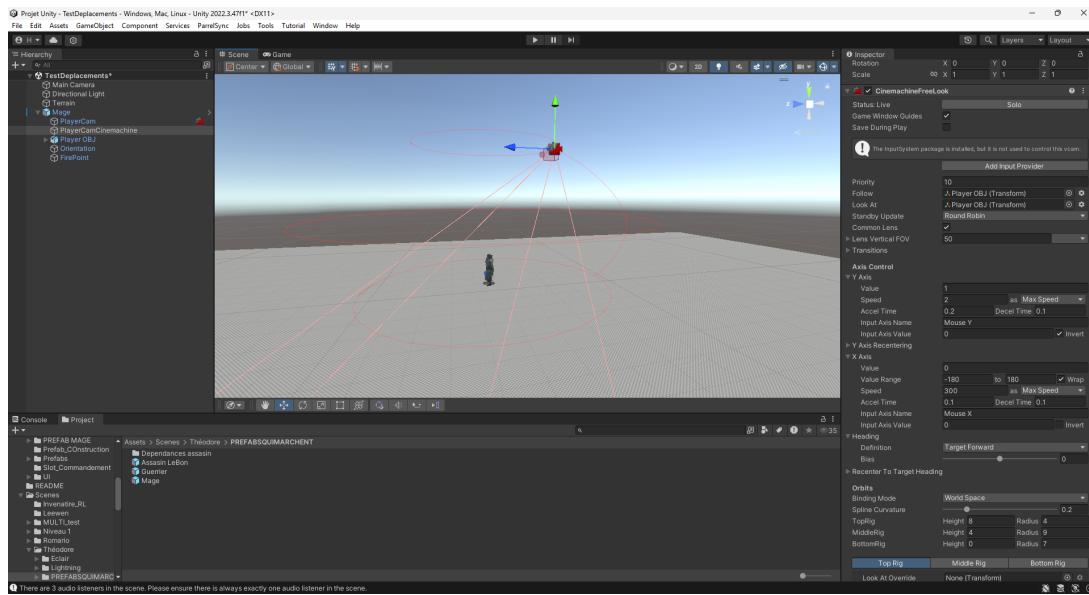


FIGURE 3 – Configuration de la caméra avec Cinemachine. Les cercles rouges définissent le champ d'action de la caméra.

Comme illustré dans la figure 2, les trois cercles rouges indiquent le champ d'action de la caméra. En ajustant leur taille, il est possible de modifier la distance et la hauteur de la caméra par rapport au joueur, ce qui permet une personnalisation fine de l'expérience utilisateur.

9.3 Les Animations

Pour les animations de tous les modèles humanoïdes, nous utilisons Mixamo, un outil proposé par Adobe qui permet de réaliser automatiquement le "rigging" des modèles. Mixamo offre également une vaste bibliothèque d'animations gratuites, que nous pouvons appliquer à nos propres modèles.

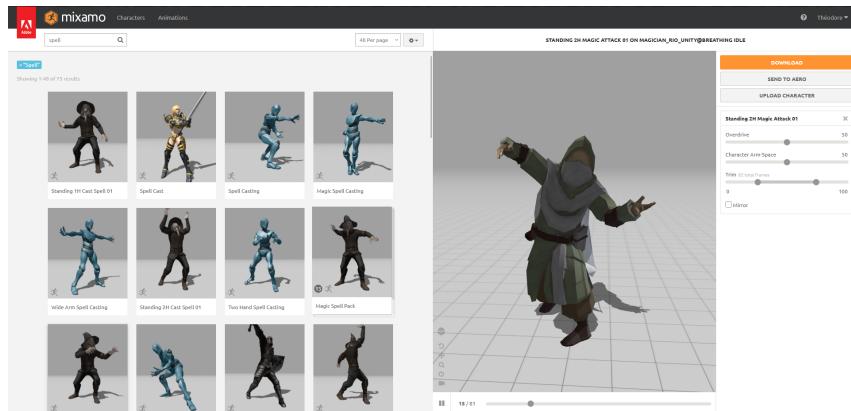


FIGURE 4 – Ci-Dessus, le logiciel d'auto-rigging Mixamo et sa bibliothèque d'animations

Comme illustré dans la figure 4, après avoir importé le modèle voulu dans Mixamo, il suffit de sélectionner l'animation voulue et de la télécharger.

Une fois les fichiers d'animation importés dans Unity, c'est l'Animator qui prend en charge la gestion des transitions entre les différentes animations, en fonction de variables booléennes. Par exemple, pour déclencher l'animation de marche, lorsque le joueur appuie sur la touche Z, la variable booléenne IsWalking passe à true, et inversement lorsque la touche est relâchée.

L'Animator se présente sous la forme d'un graphe, où chaque nœud représente une animation distincte pour un objet. Les transitions entre ces animations sont créées en reliant les nœuds entre eux. Pour assurer des transitions fluides, des effets de fondu sont appliqués.

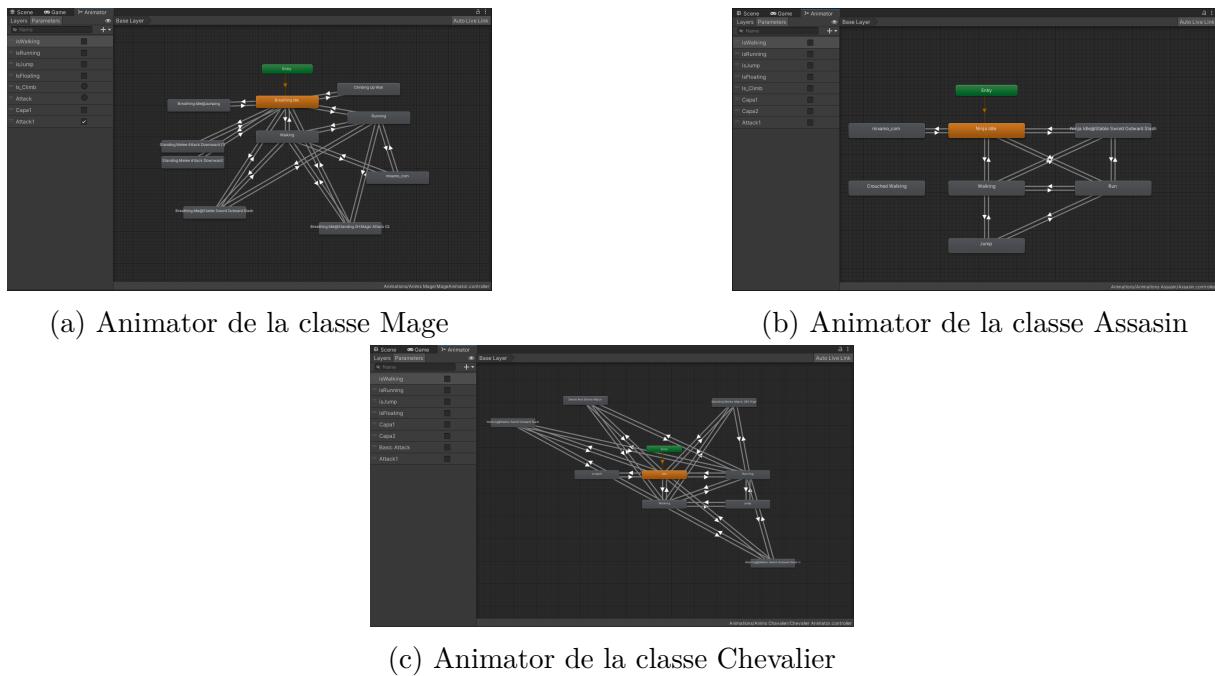


FIGURE 5 – Illustration des trois animators

Comme illustré dans la figure 5, les *Animator* des trois classes du jeu gèrent toutes les animations, des déplacements aux attaques.

10 Les Classes

Dans notre projet, nous avons structuré notre code autour de différentes classes C#, que nous avons organisées en deux grandes catégories : les classes utilitaires, qui centralisent les fonctionnalités de base, et les classes jouables, qui définissent les personnages incarnés par les joueurs.

10.1 Les Classes C# Utilitaires

La Classe PlayerMouvement Cette classe gère les déplacements des personnages. Elle inclut la gestion des entrées utilisateur, le contrôle de la vitesse de déplacement et l'intégration aux animations. Elle est pensée pour être réutilisée : toutes les classes jouables héritent de `PlayerMouvement`, évitant ainsi la duplication de code.

La Classe BasePlayer `BasePlayer` est une classe mère conçue pour regrouper toutes les fonctionnalités communes aux différentes classes jouables. Elle hérite de `PlayerMouvement` pour intégrer directement les capacités de déplacement. Elle initialise également les caractéristiques de base des personnages (points de vie, dégâts, vitesse, etc.), qui varieront selon la classe jouée.

Fonctionnalités de BasePlayer

- **Gestion des points de vie :**
 - Méthode pour infliger des dégâts et détecter la mort du joueur.
 - Méthode pour soigner le joueur.
- **Respawn** : Réapparition du joueur à un point défini.
- **Gestion de l'expérience et du niveau :**
 - Ajout d'expérience.
 - Passage de niveau avec amélioration des statistiques.

Avantages de cette structure Regrouper ces fonctionnalités dans `BasePlayer` permet une meilleure organisation du code, une forte réutilisabilité, et une facilité d'ajout de nouvelles classes jouables.

10.2 Les Classes Jouables

Les classes jouables héritent de `BasePlayer` et représentent les différents types de personnages que les joueurs peuvent incarner. Chaque classe jouable redéfinit ou étend certains comportements (par exemple, les capacités spéciales ou les statistiques de départ). Le joueur peut choisir la classe qu'il va jouer entre trois différentes classes.

Le Mage La classe "Mage" permet au joueur d'utiliser deux capacités spéciales : lancer une boule de feu et faire tomber la foudre où bon lui semble.



(a) Capacité 1 : Boule de Feu



(b) Capacité 2 : Foudre

FIGURE 6 – Illustration des capacités spéciales du Mage

Classe Guerrier En choisissant la classe "Guerrier", le joueur incarne un chevalier en armure. Cela lui permet d'avoir accès à deux coups d'épée spéciaux. Sauter en lancant un grand coup d'épée devant lui et une attaque à 360 degrés.



(a) Capacité 1 : Saut



(b) Capacité 2 : 360°

FIGURE 7 – Illustration des capacités spéciales du Chevalier

Classe Assassin En choisissant la classe "Assassin", le joueur incarne un Assassin, personnage mystérieux et discret. Cette classe permet au joueur de devenir invisible et s'échapper en laissant de la fumée derrière lui.



(a) Capacité 1 : Invisibilité



(b) Capacité 2 : Dash

FIGURE 8 – Illustration des capacités spéciales de l'Assassin

11 Les VFX et particules

11.1 Les particles systems

Les **particles systems** sont le moyen le plus basique de faire des particules dans unity. Nous avons utilisé des particles systems à plusieurs reprises.



(a) Dans la trainée de la boule de feu



(b) Flammes de l'arène de multijoueur

FIGURE 9 – Ci-dessus les exemples d'utilisation des particles systems

11.2 Les VFX Graphs

Les VFX graphs sont un moyen beaucoup plus avancé de faire des particules et des effets visuels. Ils se présentent, comme leur nom le laisse entendre sous forme de graphes. Nous avons utilisé les VFX graphes à de maintes reprises, notamment pour la capacité de la classe "Mage" qui lui permet de faire tomber la foudre. En effet il a fallut utiliser un VFX graph pour l'éclair en lui-même mais aussi pour les particules situées au lieu d'impact de l'éclair.

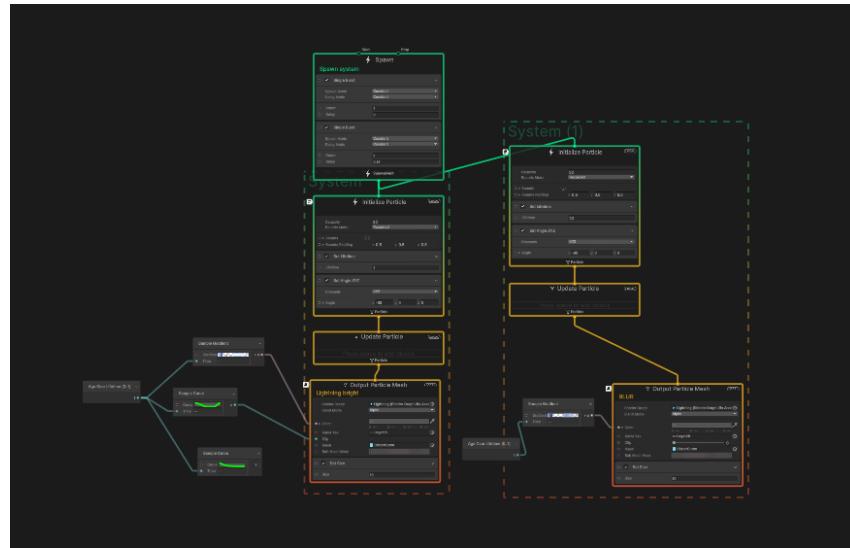


FIGURE 10 – Le VFX graph de l'éclair

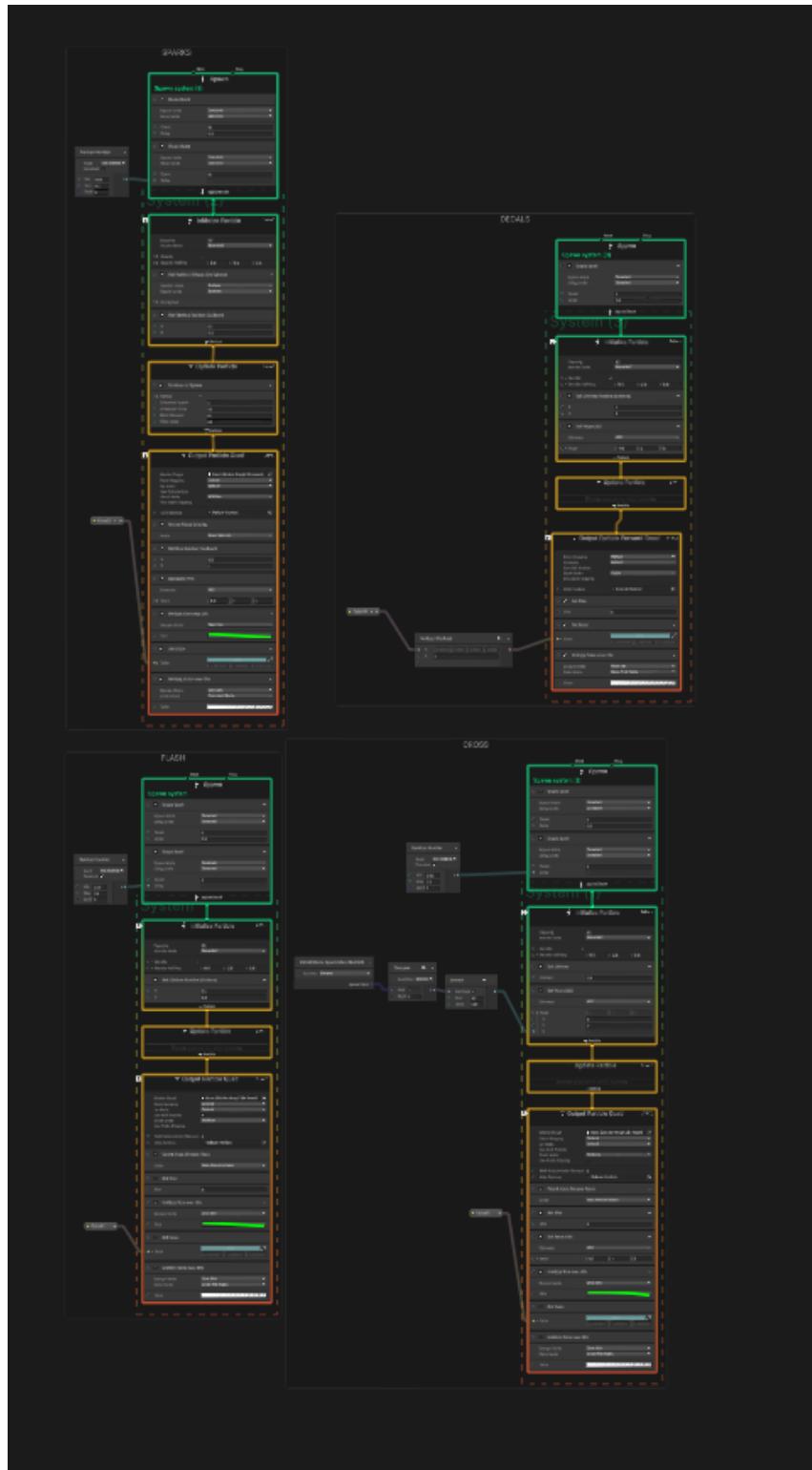


FIGURE 11 – Le VFX graph de l’impact

12 Système d’attaque

12.1 Objectifs

Le système d’attaque devrait permettre :

- D'équiper et déséquiper une arme.
- D'exécuter des attaques comme des coups de pied ou de poing sans arme, ou des coups spécifiques avec une arme.

12.2 Conception

12.2.1 Équiper/déséquiper une arme

Pour assurer que l'arme suit les mouvements du personnage, nous avons utilisé :

- Un *Rigidbody* pour les collisions.
- Un *fixepoint* sur la main du personnage.

L'arme est initialement attachée à une position de base (ex. : le dos). Lorsqu'une touche est pressée, elle devient enfant de la main du personnage avec la méthode `weapon.SetParent()`.

12.2.2 Gestion des attaques

Les attaques sont gérées par un *Animator Controller* relié à un script. Les étapes sont :

1. L'état initial est *Idle*.
2. Les animations d'attaque sont reliées à cet état via des *Triggers*.
3. Le script active les *Triggers* selon les actions du joueur :
 - Arme équipée : clic droit → coup d'épée.
 - Arme non équipée : clic droit → coup de poing.

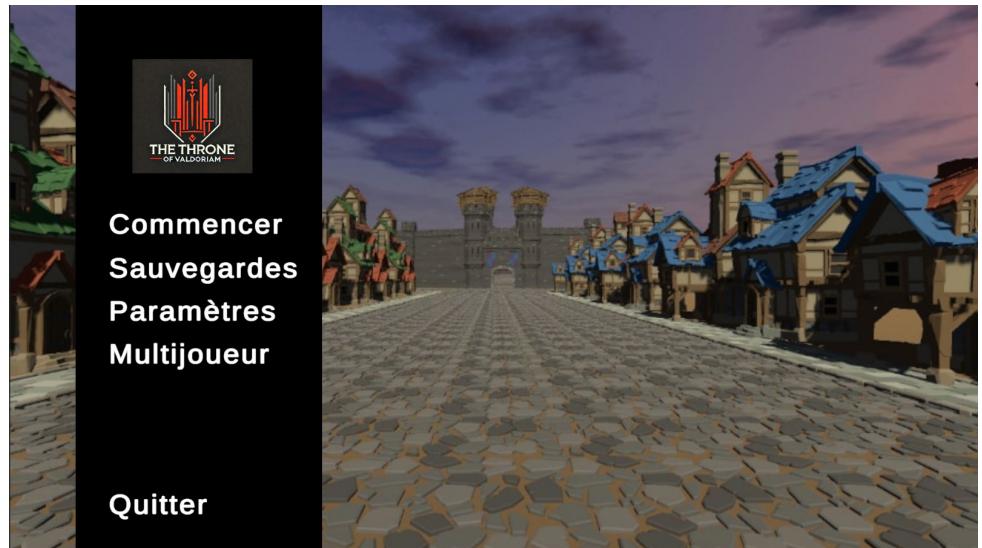
13 Interface utilisateur

13.1 Menu Principal

Le menu principal constitue le premier élément que va apercevoir le joueur lors du lancement du jeu. Il a été décidé dans un premier temps de se concentrer sur son aspect technique avant son aspect visuel, afin que le confort qu'apporte celui-ci se retrouve dans sa navigation et ses fonctionnalités tout comme dans son style esthétique.

Dans un premier temps, nous retrouvons dans le menu principal cinq boutons permettant différentes actions :

- Commencer
- Sauvegardes
- Paramètres
- Multijoueur
- Quitter

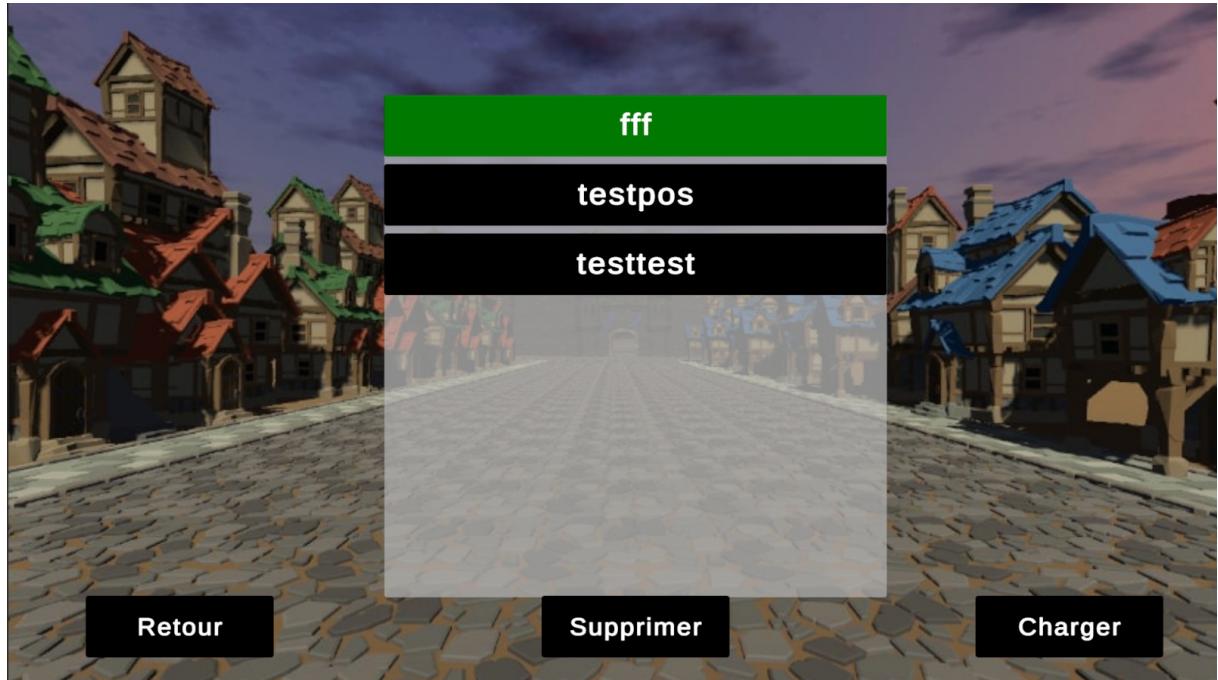


Commencer

Le bouton Commencer a pour fonction de charger la scène qui servira de premier niveau pour le joueur. Ainsi, il faut comprendre que le but de ce bouton n'est pas de continuer une partie, mais bien d'en commencer une nouvelle. Il est important dans la mesure où sans lui, il est impossible de charger la scène du niveau 1, permettant par la même occasion de pouvoir tester la compatibilité menu/jeu.

Sauvegardes

Le bouton Sauvegardes a pour objectif de répertorier les différentes sauvegardes créées par le joueur, afin de pouvoir charger ses éléments et charger la scène et les informations correspondantes.



Liste des sauvegardes

Paramètres

Le bouton Paramètres est le plus important et le plus complexe puisqu'il mène aux différents paramètres de jeu. Parmi ces paramètres, on retrouve :

- Graphismes
- Contrôles
- Son
- Retour



Ces paramètres ont la particularité d'ouvrir des boîtes de dialogue permettant aux joueurs d'interagir avec des curseurs pour régler le volume du son, un tableau déroulant pour choisir sa résolution, ou encore un ensemble de boutons permettant de changer ses touches.

L'option Graphismes permet de changer la résolution du jeu, les résultats étant observables seulement en utilisant la version téléchargeable et jouable du projet.

Le bouton Contrôles permet de pouvoir gérer l'attribution des touches aux différentes actions du personnage joueur, le bouton Son ouvre une boîte de dialogue qui invite le joueur

à choisir le volume souhaité qui évolue en même temps que le joueur déplace le curseur et enfin le bouton Retour permet de revenir à la première instance du menu principal, en sauvegardant les modifications effectuées.

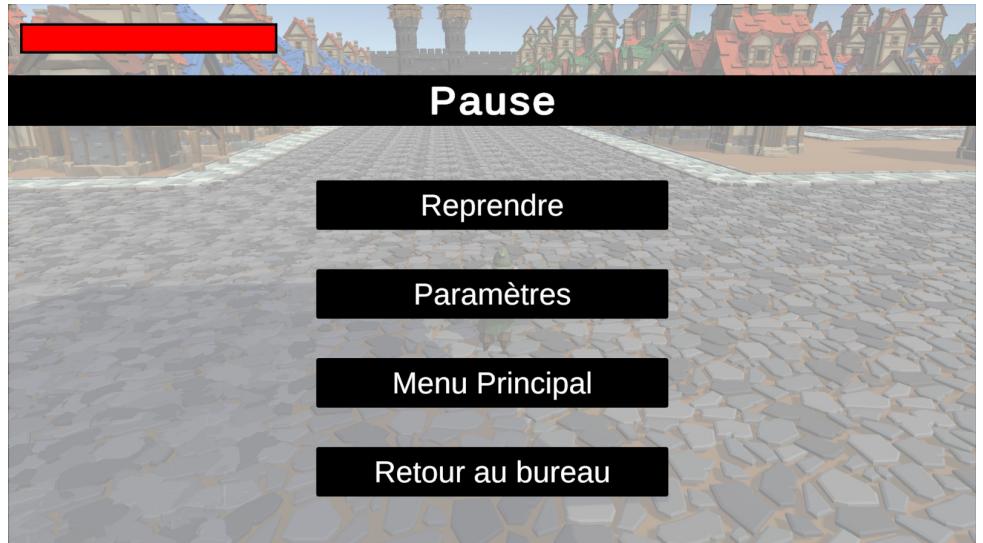
Quitter

Le bouton Quitter a la même fonction que le bouton Commencer, c'est-à-dire quitter la scène du menu principal. Ici, plus concrètement, le bouton quitte l'application, arrêtant le jeu et tout processus lié.

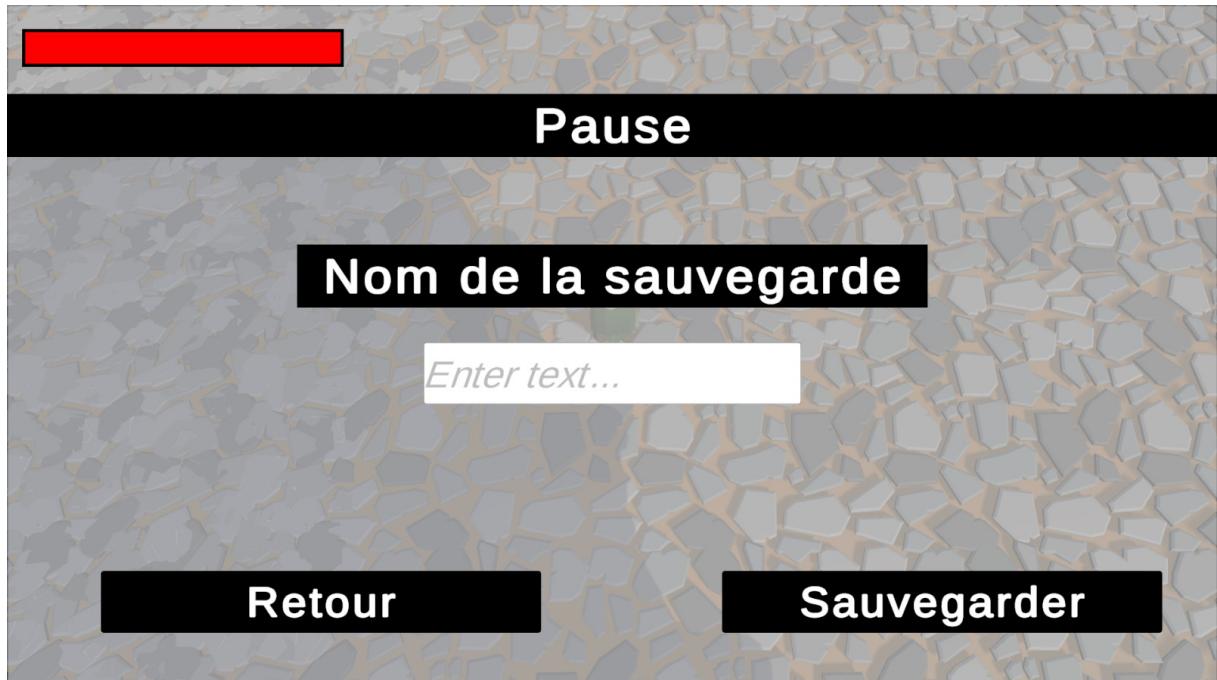
13.2 Menu Pause

Le menu pause a pour objectif de stopper le processus de jeu lorsque le joueur souhaite apporter des modifications à l'environnement de celui-ci. Parmis les modifications possibles, il y a :

- Reprendre
- Paramètres
- Menu Principal
- Retour au bureau

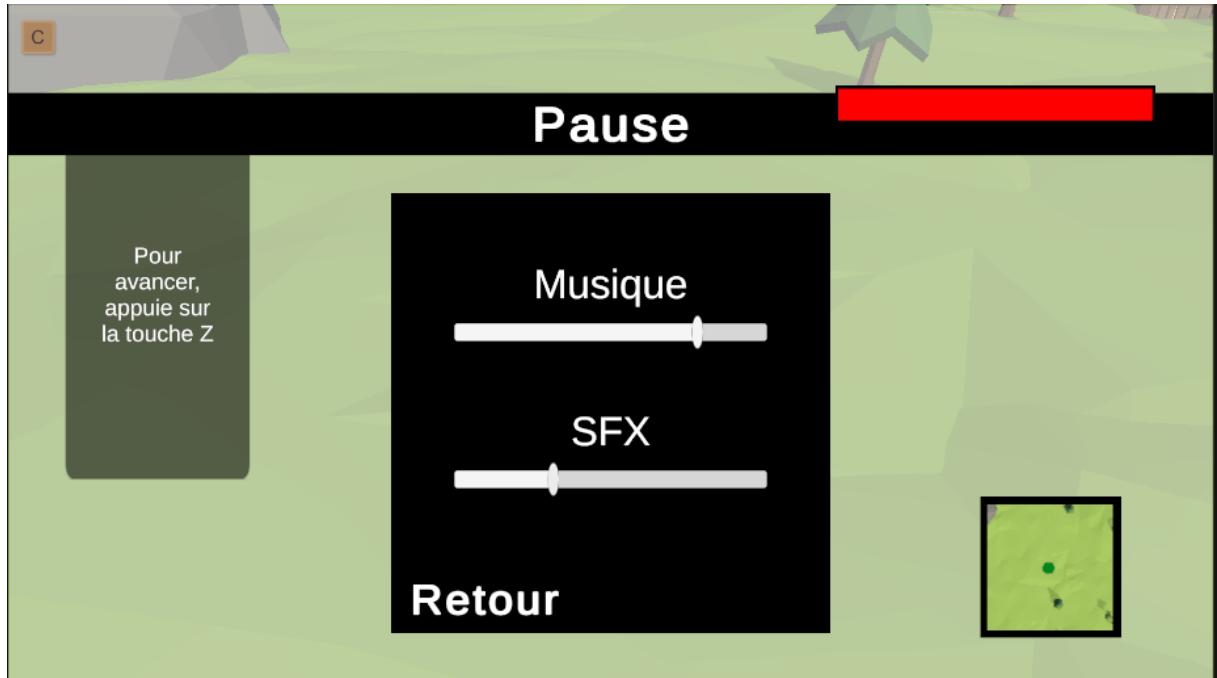


En somme, le menu pause reprend les mêmes fonctionnalités que le menu principal, pour permettre au joueur de modifier certains paramètres sans l'obliger de revenir au menu. Nous retrouvons alors le bouton "Reprendre" pour arrêter le menu pause, ce qui fonctionne aussi en appuyant sur "echap", "Paramètres" pour retrouver les mêmes options que dans le menu principal et le bouton "Retour au bureau" pour quitter le jeu.
Désormais, le bouton "Paramètres" apporte au joueur la possibilité de sauvegarder sa partie, en attribuant un nom à la sauvegarde.



Option "sauvegarder"

De plus, le menu pause contient les boîtes de dialogues permettant de modifier le son et la vfx du jeu, à travers deux curseurs correspondant à chacun :



Boîte de dialogue "Son"

Un gestionnaire dans la scène principale reprend alors dynamiquement les valeurs de ces curseurs, qui ont aussi les mêmes valeurs que depuis la scène du menu principal. Ce gestionnaire va appliquer les valeurs de ces curseurs à toutes les sources de sons dans la scène de jeu, permettant un comfort sonore fluide et constant tout au long de l'expérience.

13.3 Choix des classes jouables

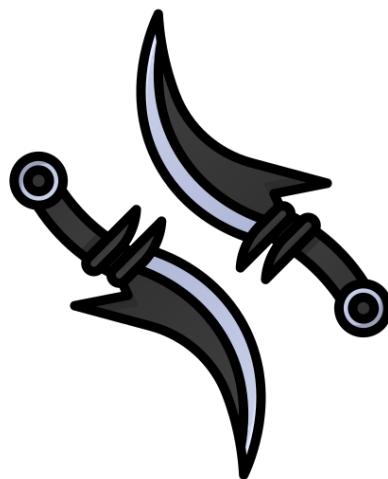
Lorsque le joueur lance une nouvelle partie, ne devons tout d'abord choisir une classe de personnage parmi les trois disponibles. Pour permettre cette sélection, nous avons développé une interface utilisateur dédiée qui apparaît seulement quand on lance une nouvelle partie ou en cas de défaite. Celle-ci se présente sous la forme d'un écran de sélection composé de trois boutons visuels, chacun représentant une classe différente à l'aide d'une image .



(a) Illustration de la classe Guerrier



(b) Illustration de la classe Mage



(c) Illustration de la classe Assassin

FIGURE 12 – Illustration des trois classes

Chaque bouton est placé de manière à être facilement identifiable et accessible dès l'ouverture du menu. Nous avons également intégré une zone de détection (hover) pour

chaque bouton, permettant d'améliorer l'interaction avec l'utilisateur. Lorsqu'un joueur survole l'un des boutons avec la souris, un effet visuel s'active afin d'afficher le nom de la classe correspondante. Cela permet non seulement de clarifier le choix pour le joueur, mais aussi de renforcer la lisibilité et l'esthétique de l'interface. Cela permet de rendre le système plus intuitif.

13.4 Difficultés rencontrées

La plus grande difficulté rencontrée lors de la création du menu pause a été l'héritage des propriétés entre le menu principal et ce menu. En effet, la modification de la touche "sauter" dans le menu principal ne garantissait pas que celle-ci soit enregistrée lors du passage au niveau 1, contenant alors le menu pause. Le même problème avait lieu dans l'autre sens. Le changement des valeurs des curseurs de son présentaient un cas de figure similaire.

Pour corriger cette erreur, nous avons utilisé la classe "PlayerPrefs" fournie par Unity, permettant d'enregistrer les informations nécessaires entre les sessions, et donc entre la scène du menu principal et le niveau 1. De cette manière, le menu pause et tous ses composants chargent les informations contenues dans la classe "PlayerPrefs" pour que les changements soient appliqués.

Lors de l'implémentation de l'interface utilisateur pour le choix des classes, nous avons rencontré un problème technique qui a nécessité des ajustements. Le souci est survenu au moment de gérer les interactions avec les boutons de sélection de classe. Initialement, lorsque nous cliquions sur l'un des boutons, l'action ne se déclenchaît pas correctement. Après analyse, nous avons constaté que le clic provoquait un blocage du curseur de la souris, ce qui empêchait les interactions avec l'interface.

Plus précisément, le système de gestion du clic figeait le curseur immédiatement après l'action, ce qui empêchait l'exécution des fonctions associées à l'événement onClick. Par conséquent, quand le joueur voulais sélectionner une classe son curseur disparaissais , aucun joueur n'était instancier et l'interface utilisateur ne disparaissais pas . Nous avons résolu ce problème en modifiant l'ordre d'exécution des événements liés à l'interface et en désactivant temporairement le verrouillage du curseur jusqu'à ce que la sélection soit validée.

14 Site Internet

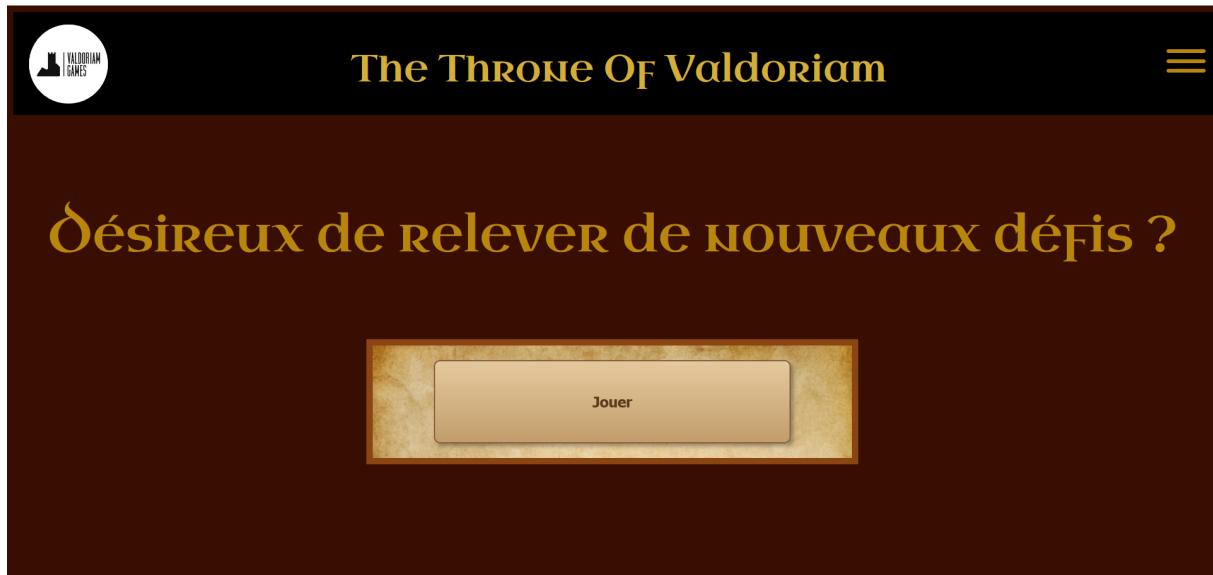
Le site représente l'image publique de Valdoriam Games, en plus des différents réseaux sociaux utilisés. Il est important dans la mesure où c'est ici que les joueurs pourront télécharger le jeu, en apprendre plus sur son histoire et celle du groupe et retrouver la bibliographie et les ressources utilisées pour le projet. le lien est le suivant : <https://bit.ly/valdoriam>

Le site comporte 6 pages différentes :

- Accueil
- A propos de Valdoriam Games
- A propos des créateurs
- Nous contacter
- Annexe

14.1 Page d'accueil

La page "Accueil" contient le bouton de téléchargement, en cours d'implémentation. De plus, un menu déroulant, disponible sur toutes les pages, permet de naviguer n'importe où.



Page d'accueil

14.2 Page "A propos de Valdoriam Games"

La page "A propos de Valdoriam Games" présente dans un court texte l'histoire du groupe, notamment la manière dont il s'est formé.

Page "A propos de Valdoriam Games"

14.3 Page "A propos des créateurs"

"A propos des créateurs" présente à travers des cartes cliquables chaque membre du groupe, accompagnées par la photo de chacun. En cliquant dessus, un court texte apparaît, appliquant un filtre à la photo pour la lisibilité du texte.

Page "A propos des créateurs"

14.4 Page "Nous contacter"

"Nous contacter" contient les liens menant à nos réseaux sociaux et à notre adresse e-mail.



Page "Nous contacter"

14.5 Page d'annexe

Enfin, la page d'annexe contient les liens importants au projet pour pouvoir accéder aux cahiers des charges de chaque soutenance et les liens pour accéder aux outils utilisés.



Page d'annexe

15 Multijoueur

15.1 Implémentation

A l'aide de tutoriels, nous avons tenté d'implémenter le multijoueur avec plusieurs bibliothèques unity, nous avons commencé par Mirror mais malheureusement avec la comptabilité avec notre version de unity, nous n'avons pas réussi à l'implémenter.

Nous avons également tenté d'implémenter le multijoueur avec Photon Pun 2 et encore une fois, ce fut un échec, car la version est trop ancienne comparée à notre version de Unity, cela s'est passé de la même façon avec Photon Fusion.

Nous nous sommes donc tourner vers Unity Netcode, avec un système d'hôte et de client qui s'échangent des informations entre eux., Nous avons réussi à dissocier les mouvements de chacun des joueurs et envoyer des informations de mouvements à l'autre joueur ce qui n'était pas le cas avant, les joueur peuvent en rejoindre un autre avec cette extension Unity qui nous est grandement bénéfique et qui nous sert à étendre notre portée au-delà du même wi-fi en avec un système de serveur que plusieurs joueurs pourront rejoindre en même temps.

Cet outil nous aide principalement à dirigé et mettre un des joueurs en tant que hôte afin que l'autre joueur (le client) rejoigne la partie.

15.2 Crédit et synchronisation

Pour linstanciation de terrain, nous avons mis un terrain plat auquel nous avons ajouté le composant network object pour qu'il soit reconnu en tant qu'objet lorsque nous créons le terrain, cela marche ainsi pour Unity Netcode.

Un des joueurs doit se mettre en tant que hôte et permettre aux autres joueurs de rejoindre la partie, tout se fait automatiquement en cliquant sur le bouton host button, l'autre doit rejoindre l'hôte en mettant l'adresse ipv4 de ce même hôte puis appuyer sur la touche entrée du clavier et cliquer simplement sur le bouton rejoindre.

Pour créer une ambiance immersive tout en optimisant nos ressources, nous avons choisi de réutiliser l'environnement du château présent dans le niveau 1 de notre mode histoire (solo). Cette décision nous a permis de proposer une petite arène nichée entre les mêmes remparts majestueux, offrant ainsi aux joueurs un cadre à la fois familier et parfaitement adapté à des combats épiques. Les pierres anciennes des murailles et les bannières flottant au vent créent une atmosphère médiévale qui renforce l'intensité des duels.



(a) Menu sans entrer d'adresse ip

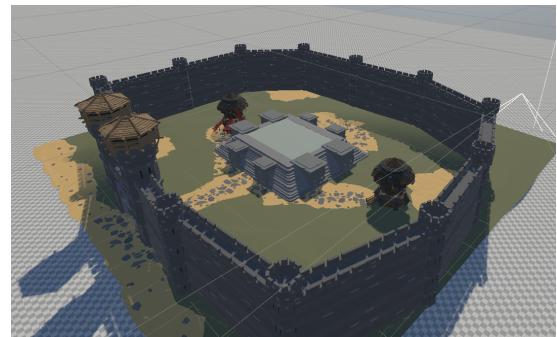


(b) menu en entrant une adresse ip

FIGURE 13 – menu principal



(a) carte vue de face



(b) carte en vue 3/4

FIGURE 14 – carte multijoueur

15.3 Mode de jeu Multijoueur

Lors de la conception de notre mode multijoueur, notre première inspiration s'est naturellement tournée vers un affrontement en joueur contre joueur, un classique indémodable lorsqu'on imagine une expérience multijoueur captivante. Nous avons imaginé un système simple mais intense : deux valeureux chevaliers s'affrontant dans un duel à l'épée, où la maîtrise des coups et des parades fait toute la différence.

Dès que deux joueurs entrent dans l'arène, le combat peut commencer immédiatement, sans temps d'attente. Nous avons délibérément opté pour un format sans fin, avec des vies illimitées pour les deux adversaires. Ce choix permet aux joueurs de se concentrer sur le plaisir pur du combat, sans la pression d'un chronomètre ou d'un score à atteindre. L'expérience est avant tout axée sur l'amusement et la compétition amicale : les duels peuvent durer aussi longtemps que les participants le souhaitent, et la partie se termine simplement lorsque l'un des joueurs – ou les deux – décide de mettre fin à l'affrontement. Ce mode multijoueur, bien que simple dans son concept, offre une rejouabilité infinie et met l'accent sur l'habileté et le timing, tout en maintenant une ambiance conviviale et décontractée. Il s'agit d'une excellente manière pour les joueurs de tester leurs compétences entre amis dans un cadre visuellement cohérent avec l'univers de notre jeu.

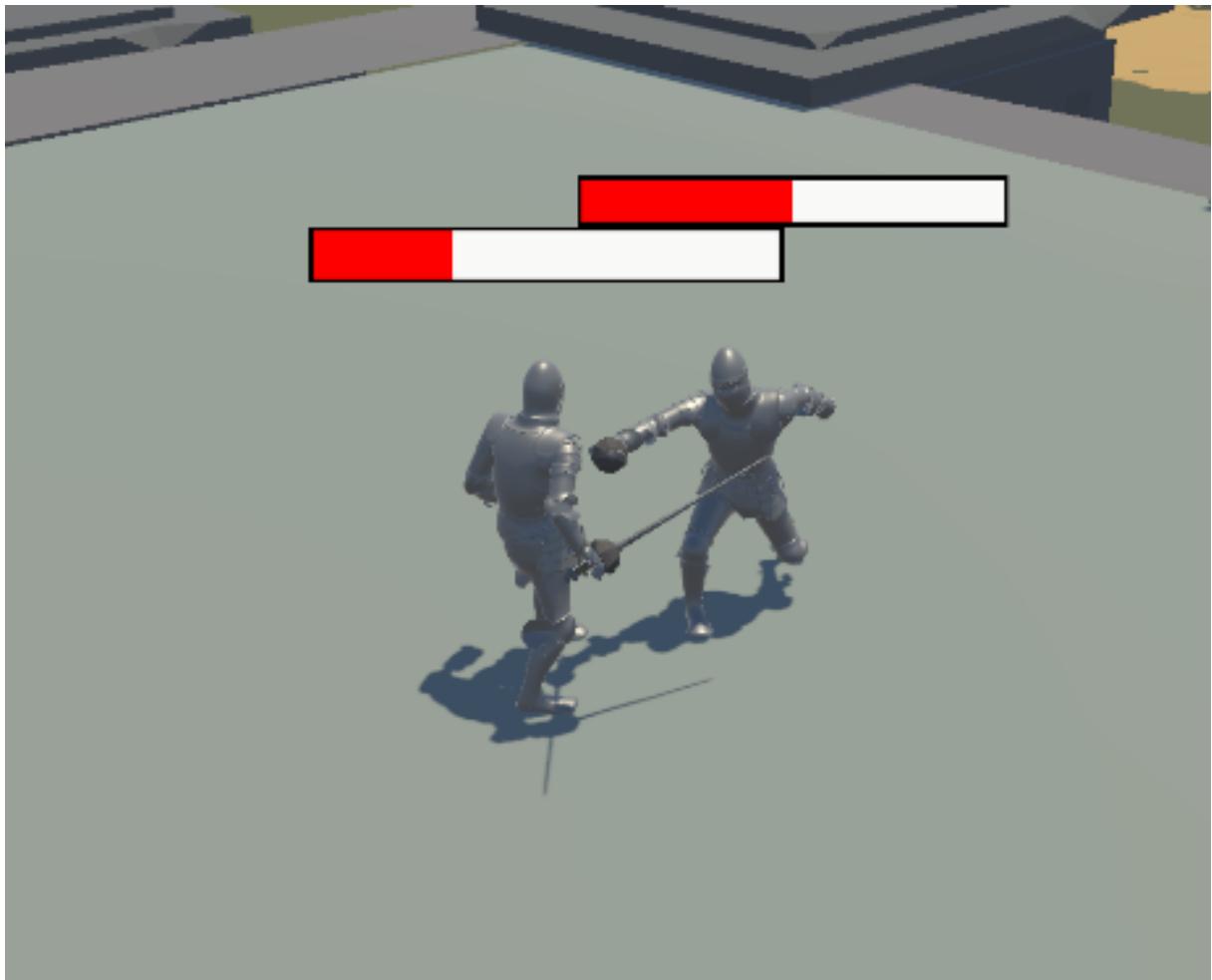


FIGURE 15 – 2 chevaliers s'affrontant

15.4 Difficultés

Durant la programmation de notre mode multijoueur, nous avons été confrontés à plusieurs difficultés techniques majeures qui ont considérablement ralenti notre progression.

La première problématique concernait la connexion à l'hôte, qui échouait aléatoirement sans raison apparente, entraînant des déconnexions forcées. Après une analyse approfondie, nous avons identifié que ce problème provenait à la fois d'un défaut dans le prefab du joueur et d'une erreur dans la scène. De plus, notre outil de test local (ParrelSync) ne mettait pas correctement à jour le clone, ce qui générait des incohérences et aggravait ces problèmes de connexion.

Ensuite, nous avons dû faire face à une désynchronisation des déplacements. Au lieu de bouger indépendamment, tous les joueurs se déplaçaient simultanément, même lorsque seul l'un d'entre eux effectuait une action. Pour résoudre ce bug, nous avons modifié notre code afin qu'il ignore les inputs des autres joueurs et n'exécute que les mouvements du personnage local, éliminant ainsi les conflits.

Enfin, nous avons découvert une limitation majeure d'Unity Netcode : l'absence de synchronisation des animations côté client. Par défaut, seuls les déplacements de l'hôte étaient visibles, tandis que les animations des autres joueurs restaient figées. Cette restriction était due à une mesure de sécurité considérant le client comme un

tricheur potentiel. Pour contourner cette contrainte, nous avons dû modifier l'Animator intégré à Unity Netcode, permettant ainsi une synchronisation complète des mouvements et garantissant que chaque joueur puisse évoluer librement tout en étant visible par les autres.

Ces défis, bien que complexes, nous ont permis d'améliorer significativement notre compréhension des réseaux dans Unity et d'optimiser notre approche du développement multijoueur.

Lors de l'implémentation de cette interface utilisateur pour le choix des classes , nous avons rencontré un problème technique qui a nécessité des ajustements. Le premier souci est survenu au moment de gérer les interactions avec les boutons de sélection de classe. Initialement, lorsque nous cliquions sur l'un des boutons, l'action ne se déclencha pas correctement. Après analyse, nous avons constaté que le clic provoquait un blocage du curseur de la souris, ce qui empêchait les interactions avec l'interface.

Plus précisément, le système de gestion du clic figeait le curseur immédiatement après l'action, ce qui empêchait l'exécution des fonctions associées à l'événement onClick. Par conséquent, quand le joueur voulais sélectionner une classe son curseur disparaissais , aucun joueur n'était instancier et l'interface utilisateur ne disparaissais pas . Nous avons résolu ce problème en révisant l'ordre d'exécution des événements liés à l'interface et en désactivant temporairement le verrouillage du curseur jusqu'à ce que la sélection soit validée.

16 IA

s natifs d'Unity tels que NavMeshAgent pour le déplacement, Animator pour l'animation, ainsi que des systèmes de détection basés sur des sphères (OverlapSphere) et la gestion des couches (Layers).

16.1 IA ennemis



FIGURE 16 – Modèle 3D ennemis

16.1.1 États comportementaux

Les IA ennemis sont structurées autour d'une **machine à états** composée de trois phases principales : Patrouille, Suspicion et Attaque.

État de Patrouille : L'IA se déplace automatiquement entre plusieurs points prédéfinis de la scène. Ces points sont parcourus en boucle, simulant un comportement de ronde. Cette phase est passive, et l'IA ne recherche pas activement de cibles.

État de Suspicion : Cet état transitoire est déclenché par un événement sonore ou visuel.

- **Détection sonore** : Le joueur génère une “zone de son” autour de lui (représentée par une sphère invisible). Si une IA ennemie entre dans cette zone, elle reçoit la position du joueur comme un point de suspicion et s'y rend.
- **Détection visuelle** : L'IA peut aussi détecter le joueur à l'œil nu, selon un champ de vision défini. Si le joueur est vu, sa position actuelle est enregistrée comme point de suspicion.

Une fois au point de suspicion, l'IA attend quelques secondes. Si elle ne détecte ni joueur ni allié, elle retourne en patrouille. En revanche, si une cible est repérée, l'IA passe en mode Attaque.

État d'Attaque Trois cas sont possibles :

- **Cible à portée** : L'IA déclenche une animation d'attaque via le système Animator.
- **Cible visible mais éloignée** : L'IA poursuit la cible en mettant à jour sa destination.
- **Cible hors de portée prolongée** : L'IA abandonne la poursuite et retourne en patrouille.

16.1.2 Animation des IA ennemis

Les animations sont contrôlées par un système **Blend Tree**, intégré dans **Animator**. Celui-ci permet une transition fluide entre plusieurs animations (comme la marche et la course) en fonction de la vitesse réelle du **NavMeshAgent**.

L'attaque est gérée séparément par un paramètre de type **Trigger**, déclenchant l'animation appropriée lorsqu'une cible est à portée.

16.1.3 Détection visuelle

La détection visuelle suit les étapes suivantes :

- **Détection par zone** : Une sphère centrée sur l'IA (via **OverlapSphere**) repère les objets proches.
- **Filtrage par angle de vue** : Seules les cibles dans un angle de 90 degrés devant l'IA sont retenues.
- **Vérification de la distance** : La cible doit être suffisamment proche.
- **Filtrage par Layer** : L'IA ne tient compte que des objets sur certaines couches (par exemple "Joueur" ou "Allié").

Si plusieurs cibles sont détectées, l'IA attaque en priorité la plus proche.

16.2 IA alliées

Les IA alliées de *The Valdoriam Game* ne disposent pas d'une intelligence autonome. Leur comportement est entièrement dicté par les ordres du joueur à travers une interface de commandement. Cette architecture permet une approche tactique du jeu, sans complexité inutile du côté du système d'IA.

16.2.1 Structure des données

ScriptableObject Allies2 Chaque allié est défini comme une instance de **ScriptableObject**, ce qui permet de séparer les données (nom, description, visuel, prefab) du comportement :

- **name** : nom de l'allié.
- **description** : courte description textuelle.
- **visuel** : sprite utilisé dans l'interface de commandement.
- **prefab** : objet 3D instancié en jeu.

Classe Allies2_Scrpit Un script attaché à chaque **GameObject** d'allié, contenant une référence à l'objet **Allies2**.

16.2.2 Gestion centrale : Manger_Commandement

Ce script singleton est le cœur du système de commandement :

- Il gère la liste des alliés actifs.
- Il affiche l'interface de commandement.
- Il relaie les ordres du joueur aux alliés via des boutons.
- Il fournit une fonction de placement d'objectifs sur la carte.

16.2.3 Ordres disponibles

1. **Recruter** : L'allié est proche et dans le bon Layer (**Allies**), le joueur appuie sur R. L'objet est ajouté à la liste d'alliés.
2. **Virer** : Si l'allié est proche et présent dans la liste, le joueur appuie sur V. L'allié quitte la liste.
3. **Suivre** : Le joueur donne l'ordre à l'allié sélectionné de suivre le joueur (**NavMeshAgent** actif avec une destination dynamique).
4. **Ne plus suivre** : Interrompt le suivi.
5. **Attaquer** : Active une logique de détection d'ennemis avec **OverlapSphere**. Si un ennemi est détecté, l'allié poursuit ou attaque.
6. **Annuler l'attaque** : L'allié arrête immédiatement tout comportement offensif.
7. **Déplacer** : Le joueur place un drapeau sur la carte avec les flèches du clavier. En validant avec G, l'allié se dirige vers ce point.

16.2.4 Comportement et animation : IA_Allie_Comportement

Le script **IA_Allie_Comportement** attache la logique d'exécution à chaque allié :

- **Suivre()** : L'allié suit le joueur à distance constante.
- **deplcement()** : L'allié se rend vers un point désigné.
- **Attack_Allie()** : L'allié recherche les ennemis dans un rayon donné, puis les poursuit et les attaque.
- **recrutement()** : Gère les interactions de proximité pour recruter ou virer un allié.
- **Poursuite()** : Fait tourner l'allié vers l'ennemi, puis le suit ou attaque selon la distance.

Les animations utilisent des booléens simples dans l'Animator :

- **Suivre** : activé lorsque l'allié est en déplacement.
- **Attack** : déclenchement d'une animation d'attaque par **SetTrigger**.

16.2.5 Interface de commande : Comm_Slot

Chaque slot dans l'interface correspond à un allié :

- Affiche le visuel de l'allié.
- Stocke les références à l'objet **Allies2** et au **GameObject** en scène.
- Sur clic, appelle **Open_Action2()** qui affiche les options tactiques : suivre, attaquer, se déplacer.

16.2.6 Interaction et Layers

Le système utilise intensivement les **LayerMask** pour séparer les états :

- **Allies** : IA non commandées.
- **Allies_Comm** : IA commandées activement.

Les fonctions **ChangeLayer()** permettent de changer dynamiquement les couches, afin de modifier l'interaction entre IA, joueur et environnement.

16.2.7 Détection spatiale

La détection d'alliés et d'ennemis repose sur :

- Physics.OverlapSphere pour les ennemis à portée.
- LayerMask pour filtrer les cibles valides.
- Vector3.Distance pour ajuster le comportement selon la proximité.

16.3 Analyse du système existant

Le système actuel est fonctionnel mais basique. Il repose sur :

- Une architecture simple : un système de commandes centralisé via Manger_Commandement.
- Des comportements contrôlés par booléens : suivre, attaquer, se déplacer.
- Des interactions par proximité et touches clavier (R, V, etc.).
- Une interface utilisateur minimale permettant de donner des ordres.

16.4 Limitations observées

L'analyse du code a permis d'identifier plusieurs limitations :

1. Mélange de logique et d'UI dans le même script (Manger_Commandement).
2. Utilisation de tuples (Allies2, GameObject) au lieu de classes dédiées.
3. Couplage fort entre les scripts (ex : IA_Allie_Comportement dépend directement de StartClass).
4. Multiples vérifications d'entrées dans des scripts non dédiés.
5. Pas de système d'état pour les IA alliées (state machine absente).
6. IA non évolutive et peu réactive à l'environnement.

16.5 Axes d'amélioration

16.5.1 Architecture logicielle

- **Séparation des responsabilités** : Créer des scripts dédiés pour UI, logique, et data.
- **Système d'états (State Machine)** : Implémenter une FSM (suivre, idle, attaque, se déplacer) pour les alliés.
- **Utiliser ScriptableObject + MonoBehaviour** : Lier les données de l'allié via un contrôleur central.
- **Créer une classe AllyController** pour encapsuler le comportement et les données de chaque IA.

16.5.2 Gameplay et contrôle

- **Commandes tactiques enrichies** : Ajout de formation, garde, patrouille.
- **Système de drag & drop** pour positionner les alliés dans une carte tactique.
- **Support manette** : Intégration de l'input system pour les consoles/PC.

16.5.3 IA et navigation

- **Pathfinding intelligent** : Utiliser NavMeshObstacle pour éviter les collisions entre alliés.
- **Ciblage intelligent** : Choisir l'ennemi le plus proche ou le plus faible.
- **Ajout d'un cooldown et pattern d'attaque** (esquive, attaque à distance, etc.).

16.5.4 Optimisation

- Éviter les **OverlapSphere** chaque frame : passer à un système d'événement ou coroutines.
- Pooling des objets (drapeaux, UI) pour éviter les instanciations coûteuses.
- Gestion du LOD ou désactivation des alliés hors de portée de vue.

16.5.5 Expérience utilisateur (UX)

- Retour visuel clair : flèches, effets de sélection, icônes de statut.
- Audio feedback : sons quand une IA reçoit un ordre ou attaque.
- Indications à l'écran pour les touches contextuelles (ex : Appuyez sur R pour recruter).

16.6 Perspectives d'évolution

À terme, le système d'alliés pourrait intégrer :

- Un arbre de comportements (Behaviour Tree) pour IA complexe.
- Un système d'expérience/compétence pour les alliés.
- Des rôles spécialisés (tank, archer, healer).
- Une sauvegarde dynamique de la composition d'équipe.

16.7 Résumé IA

Le système actuel pose une base fonctionnelle solide pour les alliés contrôlés. Cependant, une refonte architecturale et une amélioration des comportements permettraient de rendre le gameplay plus riche, plus fluide et plus immersif.

Le système d'alliés est un bon exemple d'architecture pilotée par le joueur, où l'IA est simple mais efficace. Les décisions sont centralisées, et la complexité de navigation, attaque ou positionnement repose sur les composants standards Unity :

- NavMeshAgent
- Animator
- Physics
- ScriptableObject

La séparation des données (via **Allies2**) et des comportements (via **IA_Allie_Comportement** et **Manger_Commandement**) permet une extensibilité simple, et l'ajout de nouveaux alliés ou de nouvelles actions est rapides.

17 Conclusion

En conclusion même si nous n'avons pas pu réaliser toutes les fonctionnalités que nous avions prévues au départ, nous sommes fiers du travail accompli et du jeu que nous avons réussi à produire.

Ce projet nous a permis de mettre en pratique nos connaissances, d'apprendre à surmonter et à résoudre des difficultés et des problèmes techniques. Cela nous a également permis de mieux comprendre le processus de développement d'un jeu vidéo.

The Throne of Valdoriam représente l'aboutissement de plusieurs mois de travail, et nous sommes satisfaits et fiers du résultat obtenu. Ce projet a été une expérience très enrichissante, qui nous a permis de progresser et de mieux appréhender les enjeux concrets de la création d'un jeu.