



VALDORIAM GAMES

2<sup>nd</sup> RAPPORT DE SOUTENANCE



# The Throne of Valdoriam

Théodore Pénicaut  
Leewen Steinmetz  
Giovanni Gast  
Romain Bailly  
Romario Laurencena

Promo 2029

13 mars 2025

## Table des matières

<b>1 Lore</b>	<b>4</b>
1.1 Cinématiques . . . . .	4
1.2 Quêtes . . . . .	4
<b>2 Mécaniques de jeu</b>	<b>5</b>
2.1 Inventaire . . . . .	5
2.2 Système de détection des objets . . . . .	5
2.3 Le système d'interface utilisateur . . . . .	6
2.3.1 La représentation graphique . . . . .	6
2.3.2 Les actions de l'inventaire . . . . .	6
<b>3 Conception de niveaux</b>	<b>7</b>
3.1 Recherche bibliographique . . . . .	7
3.2 Conception de la carte . . . . .	7
<b>4 Répartition des tâches</b>	<b>8</b>
<b>5 Avancement</b>	<b>9</b>
<b>6 Détails Techniques</b>	<b>10</b>
6.1 Déplacements et Animations . . . . .	10
6.2 Correction et mise en place de l'Animator . . . . .	10
6.3 Les Classes . . . . .	12
6.3.1 Fonctionnalités de BasePlayer . . . . .	12
6.3.2 La Classe PlayerMouvement . . . . .	12
6.3.3 Les Avantages de la Classe Mère BasePlayer . . . . .	12
<b>7 Système d'attaque</b>	<b>12</b>
7.1 Objectifs . . . . .	12
7.2 Conception . . . . .	13
7.2.1 Équiper/déséquiper une arme . . . . .	13
7.2.2 Gestion des attaques . . . . .	13
<b>8 Interface utilisateur</b>	<b>14</b>
8.1 Menu Principal . . . . .	14
8.2 Menu Pause . . . . .	16
<b>9 Site</b>	<b>18</b>
<b>10 Multijoueur</b>	<b>19</b>
10.1 Implémentation . . . . .	19
10.2 Création du terrain et synchronisation . . . . .	19
10.3 Potentiel(s) Mode(s) de jeu . . . . .	19
10.4 Difficultés . . . . .	20
<b>11 IA</b>	<b>21</b>

<b>12 Bilan</b>	<b>22</b>
12.1 Réalisés . . . . .	22
12.2 Restants . . . . .	22

## Introduction

Dans ce premier rapport de soutenance, nous allons présenter les avancées réalisées sur notre jeu **The Throne of Valdoriam** et notre ressenti lors du développement. Ce rapport contient les différentes étapes que nous avons franchies et les éléments que nous avons déjà finalisés ainsi que ce qu'il nous reste à accomplir pour mener à bien ce projet et atteindre nos objectifs finaux.

Ce document traitera principalement de l'aspect technique du jeu, avec en annexe plusieurs images pour illustrer nos propos.

# 1 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 servira de feuille de route lors de la conception du système de quêtes.

## 1.1 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.

## 1.2 Quêtes

- **Quête 1 : Tutoriel.** Le joueur apprend différentes compétences selon la classe choisie :
  - **Chevalier** : Attaquer et parer avec un bouclier.
  - **Mage** : Lancer des sorts (protection et attaque).
  - **Assassin** : Agir furtivement et tendre des pièges.Tous les joueurs apprennent également les déplacements via une course-poursuite.
- **Quête 2 : Entraînement.** Missions à réaliser :
  - Destruction d'un artefact.
  - Mission d'assassinat.
  - Combat contre le chef de la Guilde.
- **Quête 3 : Vengeance, partie 1.** Infiltration dans le centre-ville réservé aux nobles et dans le manoir du comte Vandover.
- **Quête 4 : Vengeance, partie 2.** Capture, interrogatoire et assassinat du comte Vandover.
- **Quête 5 : Exploration.** Exploration d'un sanctuaire pour découvrir les véritables origines de Nilhum. Combat contre une créature inconnue avec deux issues possibles :
  - Victoire : récompenses spéciales et un allié guide le joueur vers la cité d'Islim.
  - Défaite : l'allié sauve le joueur mais impose une quête supplémentaire (**Quête 5.5 : Punition**).
- **Quête 5.5 : Punition.** Aider l'allié en tuant des bandits dans un camp.
- **Quête 6 : Protection.** Défense de la cité d'Islim contre des trolls et des démons.
- **Quête 7 : Le Trône.** Retour à la ville de départ pour récupérer le trône de Valdoriam en éliminant les traîtres.

## 2 Mécaniques de jeu

### 2.1 Inventaire

Dans le cadre de la soutenance, nous avons décidé que je m'occuperais de la création ainsi que du développement de l'inventaire du joueur.

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 **ScriptableObject**s. 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.

### 2.2 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.

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.

## 2.3 Le système d'interface utilisateur

### 2.3.1 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 **Canaux**, 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.

### 2.3.2 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
- Consommables
- Sorts

#### Les actions possibles :

- Poser
- Détruire
- Équiper
- Consommer

Évidemment, quel que soit son type, chaque objet peut être posé ou détruit. Pour l'instant, seules ces deux actions ont été implémentées.

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.

## 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 grande variété d'ennemis, allant des simples orcs aux puissants boss qui règnent sur ces royaumes.

## 3 Conception de niveaux

### 3.1 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.

### 3.2 Conception de la carte

La carte actuelle représente Isildur, la ville de départ. Elle est divisée en trois parties principales :

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



## Mode Multijoueur

Nous prévoyons d'intégrer deux modes multijoueurs :

- **Mode JcJ** : Un combat singulier dans une arène où deux joueurs s'affrontent dans un duel intense.
- **Mode Coopératif** : Les joueurs pourront collaborer et participer à la quête principale à deux, unissant leurs forces pour surmonter les défis du jeu.

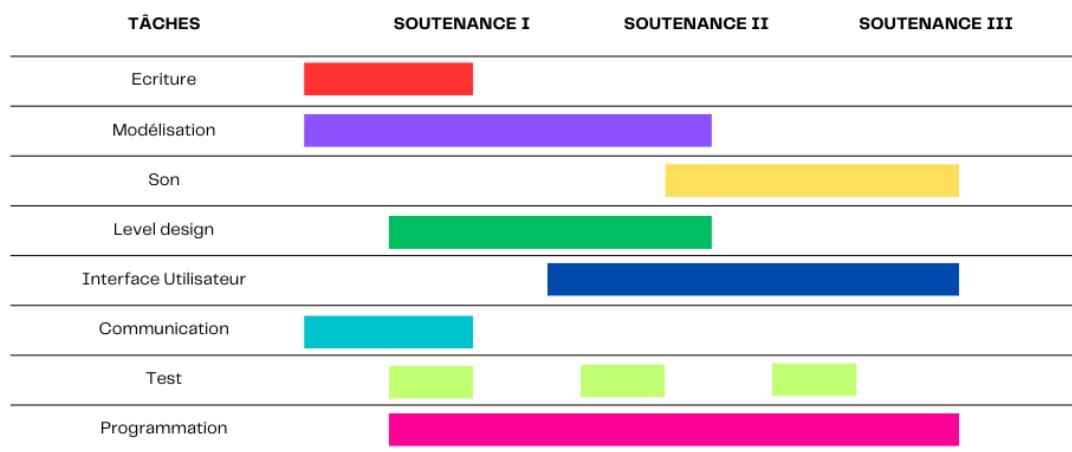
## 4 Répartition des tâches

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.

## 5 Avancement

# THE THRONE OF VALDORIAM



## 6 Détails Techniques

### 6.1 Déplacements et Animations

Après avoir fusionné nos branches respectives sur GitHub, nous avons constaté d'importants dysfonctionnements au niveau du système de déplacements et d'animations. Les animations ne se synchronisaient plus correctement avec les mouvements du joueur : elles se bloquaient presque à chaque action, ce qui nuisait à la fluidité du gameplay.

En parallèle, les déplacements étaient également affectés par des bugs. Le personnage semblait se décentrer par rapport à l'axe de rotation de la caméra, ce qui entraînait des incohérences dans les contrôles, rendant presque impossible le déplacement.

### 6.2 Correction et mise en place de l'Animator

Pour corriger ces problèmes, nous avons dû refaire tout l'*Animator* de la classe mage, la seule que nous ayons pour l'instant, l'*Animator* étant le lien entre les touches pressées et le déclenchement des animations.

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

Comme illustré dans la figure 1, l'*Animator* du personnage principal gère actuellement uniquement les animations liées aux déplacements. Nous envisageons d'ajouter d'autres animations à mesure que le développement avance.

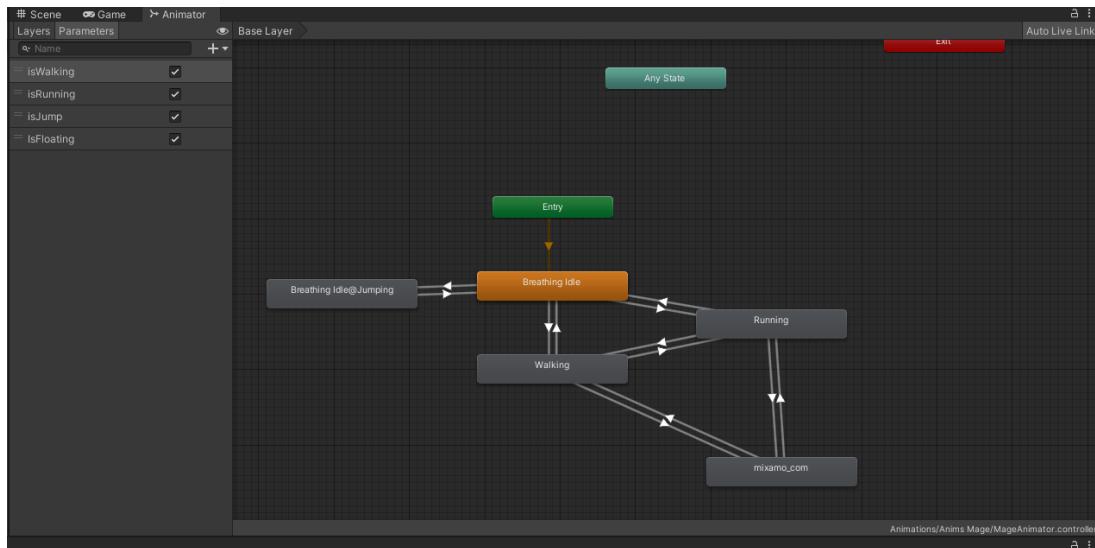
FIGURE 1 – *Animator* de la classe mage.

FIGURE 2 – Personnage de la classe mage.

## 6.3 Les Classes

Nous avons eu l'idée, pour notre jeu, de créer différentes classes jouables avec un style unique pour chacune d'entre elles. Pour cela, nous avons décidé de créer une classe mère nommée **BasePlayer**, de laquelle hériteront toutes les classes jouables.

Cette classe a pour objectif d'initialiser les différentes caractéristiques de base des personnages, comme les points de vie maximum, les dégâts, la vitesse de déplacement, etc. Ces caractéristiques varieront en fonction de la classe choisie par le joueur, permettant ainsi une diversité de gameplay.

### 6.3.1 Fonctionnalités de **BasePlayer**

La classe **BasePlayer** inclut plusieurs fonctions indispensables pour le bon fonctionnement du jeu :

- **Gestion des points de vie :**
  - Une méthode permet de modifier les points de vie du joueur lorsqu'il subit des dégâts. Si ses points de vie tombent à 0, son état passe de "vivant" à "mort".
  - Une autre méthode permet de soigner le personnage, augmentant ainsi ses points de vie.
- **Respawn :** Une méthode **Respawn** permet de faire réapparaître le personnage à un point d'apparition défini.
- **Gestion de l'expérience et du niveau :**
  - Une méthode permet de modifier l'expérience du joueur.
  - Une autre méthode augmente le niveau du joueur, ce qui a pour effet d'améliorer ses caractéristiques de base en fonction du niveau atteint.

### 6.3.2 La Classe **PlayerMouvement**

Pour gérer les déplacements des personnages, nous avions déjà une classe appelée **PlayerMouvement**. Afin de simplifier les choses et d'éviter de dupliquer le code, nous avons décidé de faire hériter la classe **BasePlayer** de **PlayerMouvement**. Cela nous permet de centraliser les fonctionnalités liées aux mouvements tout en bénéficiant des avantages de la structure héritée.

### 6.3.3 Les Avantages de la Classe Mère **BasePlayer**

Le principal avantage de la classe mère **BasePlayer** est qu'elle centralise toutes les méthodes et fonctionnalités communes aux différentes classes jouables. Cela nous permet d'écrire du code réutilisable et de faciliter l'ajout de nouvelles classes dans le futur.

## 7 Système d'attaque

### 7.1 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.

## 7.2 Conception

### 7.2.1 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()`.

### 7.2.2 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.

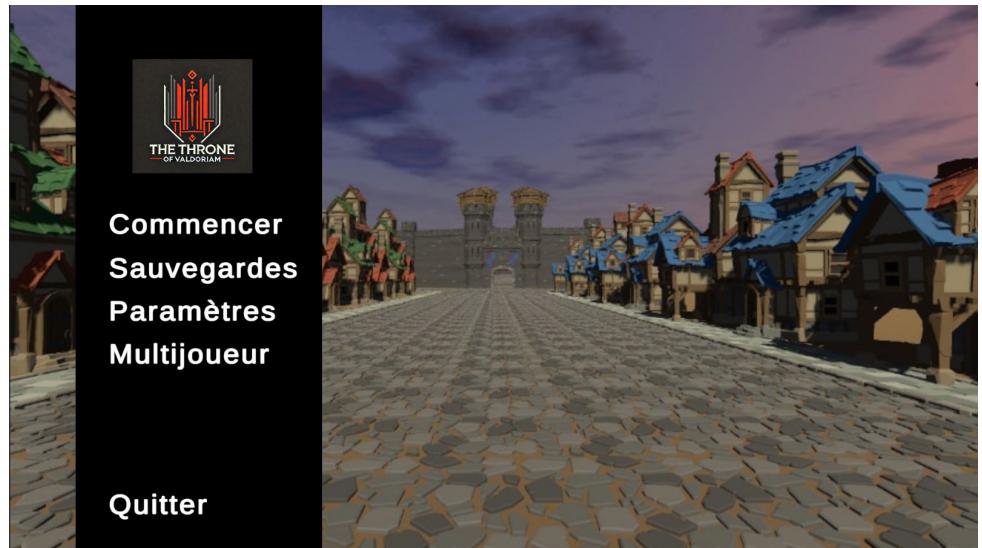
## 8 Interface utilisateur

### 8.1 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

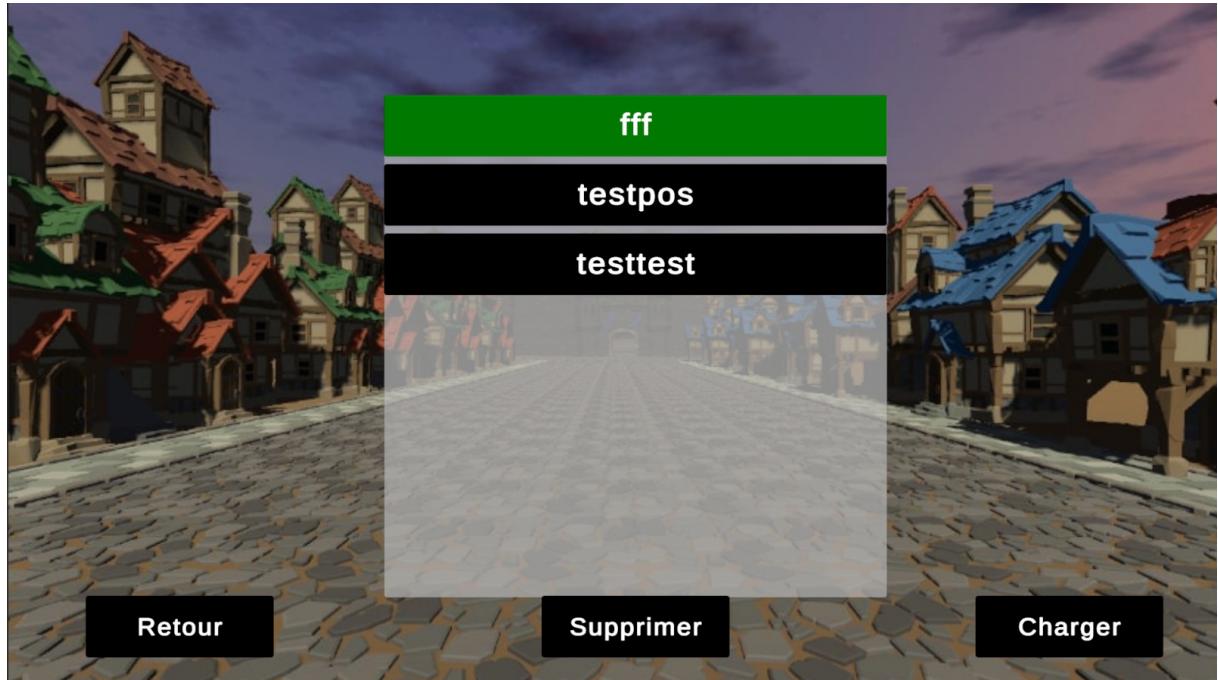


#### Commencer

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.

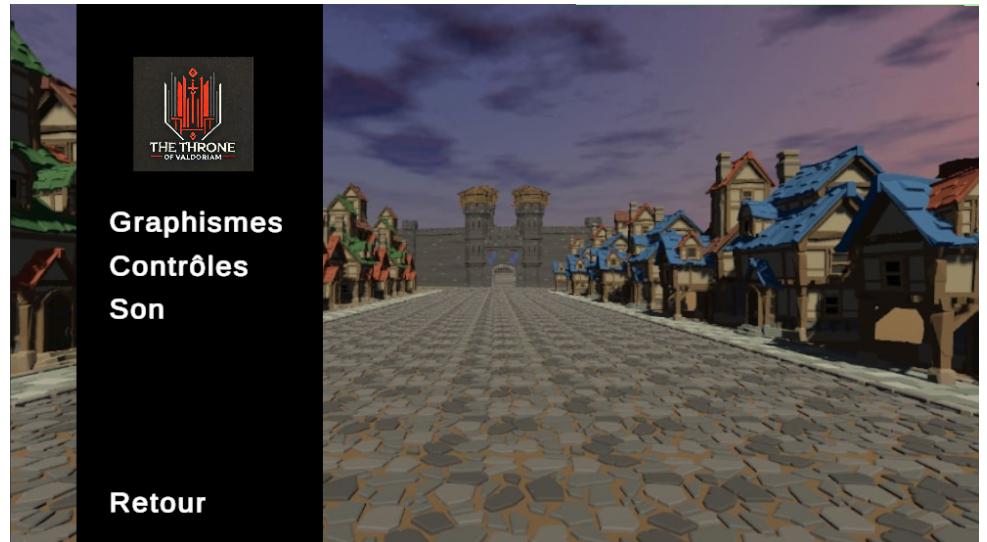


### Liste des sauvegardes

## 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 pour le moment 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.

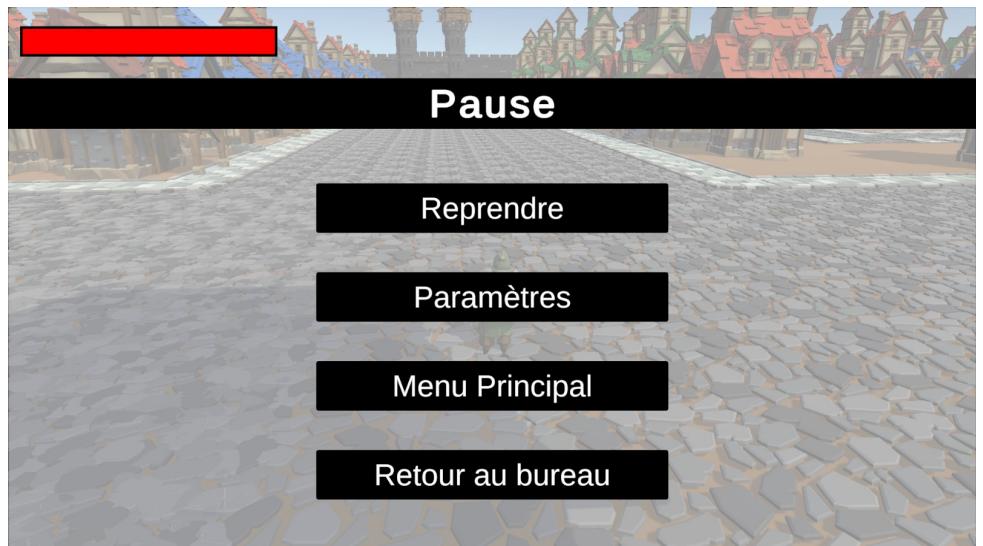
### Quitter

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.

## 8.2 Menu Pause

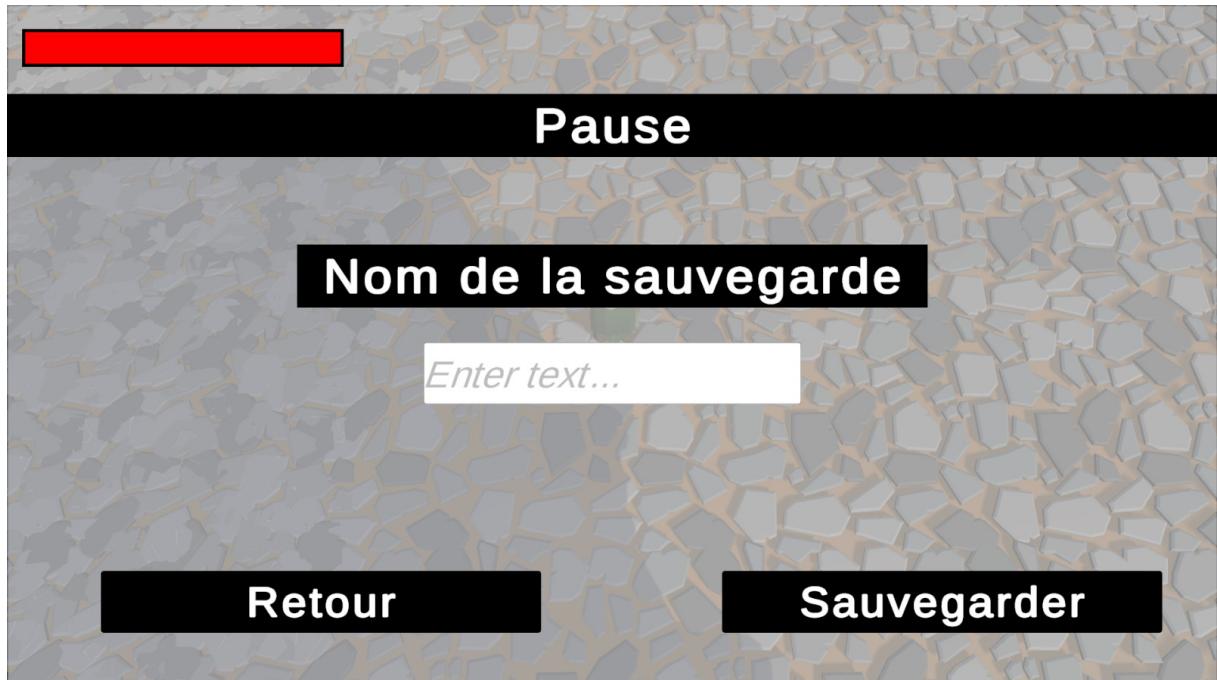
Le menu pause a pour objectif de stopper le processus de jeu lorsque le joueur souhaite apporter des modifications à l'environnement de celui-ci. Parmis les modifications possibles, il y a :

- 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"

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. 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.

## 9 Site

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 site comporte 6 pages différentes :

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

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.

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

"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.

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

## 10 Multijoueur

### 10.1 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. En ce moment, nous sommes sur Unity Netcode, avec un système d'hôte et de client qui s'échangent des informations entre eux. Pour l'instant, nous n'avons pas réussi à dissocier les mouvements de chacun des joueurs ni envoyer des informations de mouvements à l'autre joueur, néanmoins nous avons pu faire en sorte qu'un joueur en rejointe un autre avec cette extension unity qui nous est grandement bénéfique et qui nous servira à étendre notre portée au-delà du même wi-fi en implémentant un système de serveur que plusieurs joueurs pourront rejoindre.

### 10.2 Création du terrain 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. La synchronisation n'est pas encore abouti mais les deux joueurs prévus initialement sont bien chargés dans la scène, ce qui prouve la connexion entre les 2 jeu, 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 cliquant simplement sur le bouton client. Les mouvements ne sont pas dissociés entre les joueurs et les joueurs ne voient pas encore d'interactions, les joueurs sont simplement dans la même scène et voit chacun une chose différente que les autres joueurs.

### 10.3 Potentiel(s) Mode(s) de jeu

#### (a) JcJ :

Avec les connaissances du multijoueurs en tête, nous avons pensé à une première fonctionnalité, le combat contre un ou plusieurs joueurs. Cela pourrait se passer dans une arène ou un Colisée où différents guerriers peuvent se battre, la victoire s'obtiendrait en plusieurs manches gagnantes, chaque manche étant un combat contre un autre joueur. Ce mode de jeu sera prioritaire sur le mode coopératif et il sera poussé si nous pensons que nous n'avons pas assez de temps pour implémenter le deuxième mode de jeu.

#### (b) Coopératif :

La deuxième fonctionnalité à laquelle nous avons pensé est le combat de plusieurs joueurs contre des vagues successives de divers ennemis présent

dans le mode histoire, duquel la dernière vague serait l'un des boss que nous aurions découverts dans le mode histoire.

Ceci sera le mode de jeu additionnel si nous avons déjà fini le premier mode de jeu multijoueur et sera assez simple à mettre en place au vu du fait que la plupart des aspects techniques et difficiles seront déjà réalisés dans le mode JcJ.

## 10.4 Difficultés

Lors de notre première soutenance, notre implémentation du mode multijoueur était encore très limitée. Nous avions juste réussi à mettre en place l'interface utilisateur de Photon Fusion. Aujourd'hui, nous avons progressé et réussi à établir une connexion permettant à plusieurs joueurs d'interagir en temps réel. Cependant, plusieurs problèmes restent à résoudre.

Tout d'abord, nous rencontrons toujours des difficultés pour dissocier les mouvements des deux joueurs. Actuellement, lorsqu'un joueur clique pour se déplacer, cela affecte encore les deux joueurs en même temps, ce qui nuit à l'expérience de jeu.

Nous devons trouver une solution pour que chaque joueur puisse se déplacer indépendamment, sans interférer avec l'autre.

Enfin, un autre problème majeur concerne la synchronisation des animations des mouvements. Actuellement, chaque joueur voit des animations différentes, ce qui crée un décalage et une incohérence visuelle. Il est essentiel de synchroniser ces animations pour que chaque action soit correctement affichée en temps réel sur tous les écrans des joueurs connectés.

Malgré ces difficultés, nous avons déjà accompli des progrès importants, et nous continuons à travailler pour résoudre ces problèmes et améliorer l'expérience multijoueur.

## 11 IA

L'implémentation de l'IA dans Unity est relativement simple grâce au package AI Package. En effet, ce package permet de "baker" le terrain, en définissant comme statiques tous les éléments qui le composent. Cette opération permet de délimiter la zone accessible à l'IA, c'est-à-dire l'espace dans lequel elle peut se déplacer. Une fois cette étape terminée, pour faire en sorte que l'IA poursuive le joueur lorsqu'il s'en approche trop, il suffit d'utiliser la fonction Distance() pour calculer la distance entre le joueur et l'IA. Si cette distance est inférieure au rayon de détection de l'IA, cette dernière commence à suivre le joueur en utilisant la méthode SetDestination(). Lorsque l'IA se rapproche suffisamment du joueur, et que la distance devient inférieure au rayon d'attaque, l'animation d'attaque se déclenche. Par ailleurs, l'animation par défaut de l'IA est l'animation "Idle". Si l'IA poursuit le joueur, sa vitesse est augmentée et l'animation de Run s'active, synchronisée avec la vitesse de déplacement de l'IA grâce à un Blend Tree.

## 12 Bilan

### 12.1 Réalisés

- Site internet
- carte visualisable
- classe mouvement
- classe BasePlayer
- Interface du menu
- Modification des touches par défaut
- Système de sauvegarde
- Début de la modélisation de la map
- Système d'attaque
- Connexion entre joueurs

### 12.2 Restants

Nous devons maintenant nous focaliser sur le développement de l'intelligence artificielle pour notre jeu, une étape essentielle pour garantir une expérience immersive et engageante. L'IA devra être crédible dans ses comportements, réagir de manière réaliste et logique aux actions du joueur, et proposer une difficulté suffisante pour correspondre au style de jeu que nous avons choisi.

Le mode multijoueur est également un point sur lequel nous devons encore travailler pour le rendre entièrement fonctionnel.

De plus, nous avons pris du retard sur les classes, nous aurions aimé en avoir une assez avancée avant cette soutenance. C'est-à-dire une qui aurait une ou deux capacités supplémentaires et des statistiques adaptées (vitesse de déplacement, régénération automatique, force de saut ...) par rapport au joueur de base. Nous avons commencé par la classe mage, nous avons le modèle 3D et les animations, le tout unifié dans un préfab qui sera invoqué quand le joueur choisira sa classe.

Pour le système de choix de classes, nous avons fait des tests sur un projet vide et ça ne sera a priori pas une grande difficulté de changer de classe et de faire apparaître le prefab correspondant à la classe choisie.

Enfin, nous devons intégrer une fonctionnalité de dash pour enrichir les mouvements des personnages et rendre les déplacements plus dynamiques. Cette mécanique permettra au joueur d'esquiver ou de se repositionner rapidement en combat, tout en réfléchissant à d'autres améliorations pour optimiser l'expérience utilisateur.