UNIVERSITÉ DE
SHERBROOKE

Projet IFT-729

Conception temps réel

Julien Levarlet
Marius Pallard
Thomas Roux-Croteau
Thouria Ben Haddi

Encadrant :
M.Patrice Roy

12 avril 2022

1 Livrable 01

1.1 Organisation du projet

La figure 1.1 présente l'organisation du code de notre projet.

Tous les joueurs de la scène (le joueur et les IAs) vont hériter de la classe abstraite *AgentController*. Cette classe permet de centraliser le code du déplacement du joueur à partir des indications de vitesse et de direction qui seront données par les classes fille en surchargeant les méthodes *GetInputVertical* et *GetInputHorizontal*. Un attribut de cette classe est Target qui permet de connaitre la localisation de la cible pour les IAs (cet attribut est nul dans le cas du joueur, qui est lui contrôlé par les touches directionnelles).

Les classes TestManager et GameManager sont chargées de vérifier respectivement le respect des contraintes temporelles et la gestion du déroulement de partie (sortie de la zone de jeu, condition de victoire).

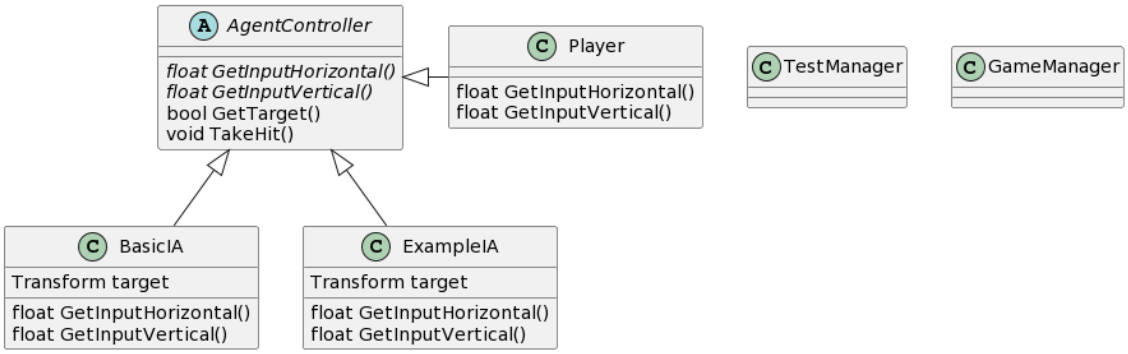

FIGURE 1.1 – Diagramme de classe de notre projet

Remarque : toutes les classes dans le diagramme UML héritent de *Monobehavior*, la classe de Unity, permettant d'attacher le code aux game-objects dans la scène.
Cela permet de surcharger les méthodes Start, Update et FixedUpdate.

Start est appelée une fois après la construction des objets dans la scène et tient un rôle proche du constructeur de la classe.
Update est appelée à chaque que rendu d'image, ce sera environ tout les 60ème de seconde dans notre cas. Elle est utilisée en général pour la gestion des inputs et de l'affichage.
FixedUpdate est appelée à chaque itération du moteur physique (dépendant de la performance de l'ordinateur. Elle est utilisée pour toutes les actions en lien avec la physique, le mouvement des joueurs dans notre cas.

1.2 Travail effectué

1.2.1 Différentes scènes

Pour tester le comportement des IAs dans différentes situations, nous avons créé plusieurs arènes plus ou moins hasardeuses pour les agents. Chaque arène présente une caractéristique différente, notament dans le type d'obstacles utilisé. Présentement, le but de ces obstacles et le design général des arènes est de tester le pathfinding de l'ennemi, c'est-à-dire s'il est capable de bien se déplacer vers le joueur.

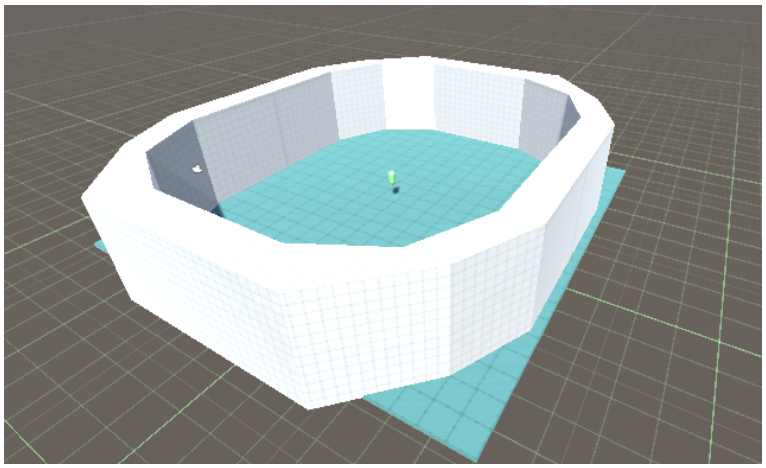

FIGURE 1.2 – Arène circulaire

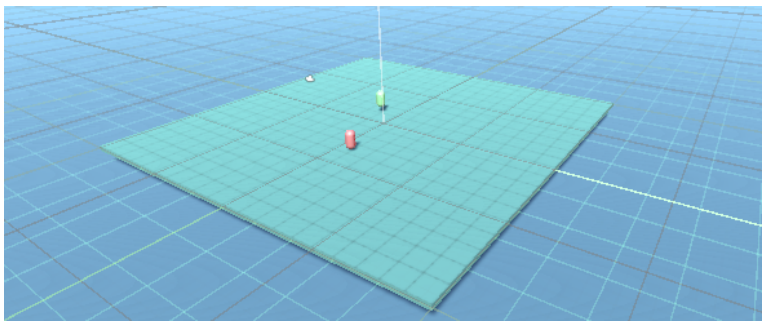

FIGURE 1.3 – Platforme simple

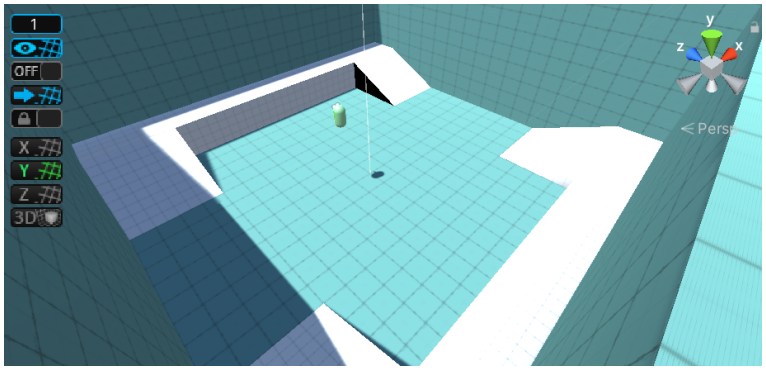

FIGURE 1.4 – Arène avec pentes

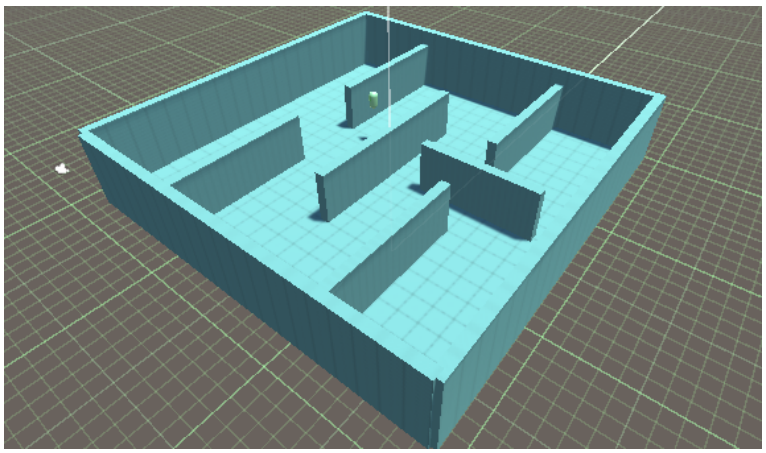

FIGURE 1.5 – Arène avec murs

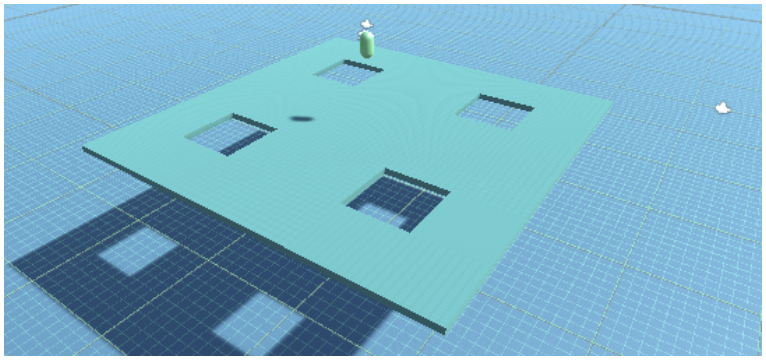

FIGURE 1.6 – Platforme avec trous

1.2.2 Cameras

Dans le but de pouvoir suivre l'action des IAs, nous avons mis en place plusieurs caméras avec des angles de vue différents dans le but de toujours pouvoir garder à l'écran ce qui nous intéresse.

Chaque caméra présente dans la scène est associée à un script déterminant son comportement. Un script *CameraManager* permet ensuite de passer d'une caméra à une autre par un appui sur la touche **TAB**. Les différents comportements implémentés sont les suivants :

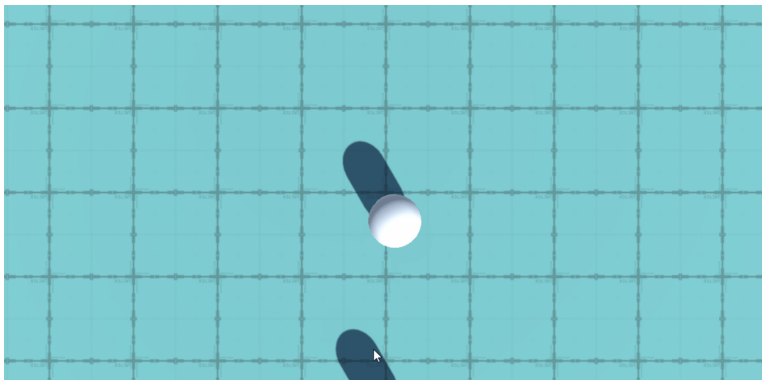

FIGURE 1.7 – Caméra en vue stratégique

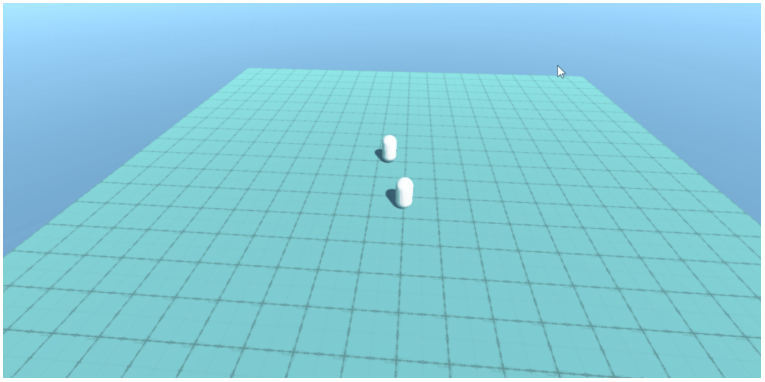

FIGURE 1.8 – Caméra à angle libre

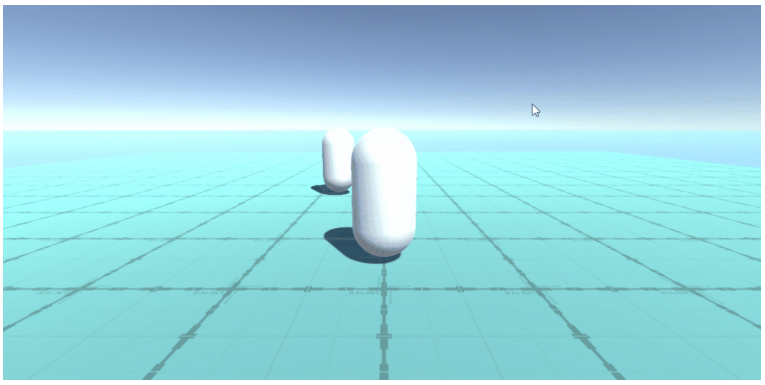

FIGURE 1.9 – Caméra de suivi

1.2.3 Mouvement des IAs

Premier test

Dans un premier temps, dans le but de tester notre implémentation, nous avons implémenté une IA ayant des mouvements aléatoires.

IA de poursuite

Une première implémentation de pathfinding est mise en œuvre à cette étape du projet, mais elle n'est pas entièrement fonctionnelle puisque l'IA s'arrête sur l'arène et devient immobile. Nous travaillons actuellement sur une méthode permettant à l'IA de patrouiller sur l'arène à la recherche du joueur.

L'IA est actuellement capable de prendre en chasse le joueur lorsqu'il se trouve à sa portée et le suit jusqu'à ce qu'il sorte de sa zone d'attaque. Actuellement cette méthode se base sur le calcul de la distance entre l'IA et le joueur, mais nous souhaiterions par la suite modifier cette approche. Nous souhaiterions mettre en place un système de détection d'objet dans un certain champ de vision, ainsi l'IA repèrerait les objets de type "Player" dans son champ de vision plutôt que de calculer des distances à chaque appel de la méthode Update().

De plus, l'IA n'est actuellement pas capable d'éviter les obstacles tels que les murs, il s'agit donc d'un point important sur lequel nous nous concentrerons une fois le pathfinding correctement implémenté.

Nous comptons actuellement un seul ennemi sur l'arène, mais prévoyons si possible d'en ajouter pour créer des niveaux de difficulté plus complexe.

1.2.4 Contraintes temps réel

Pour le moment une seule contrainte temps réel est vérifiée, c'est le rendu d'images tous les 60^{e} de seconde, vérifiable grâce à l'implémentation d'un compteur du nombre d'images par seconde à l'écran pendant le temps de jeu.

1.3 Prévisions

Dans la suite du projet, notre but est de peaufiner les éléments déjà présents, en ajoutant la vérification automatique des contraintes temps réel et l'implémentation de nouveaux algorithmes.

Dans un premier temps, l'utilisation d'un algorithme de pathfinding prenant en compte les obstacles sera étudiée de type Dijkstra ou A*.
Puis nous nous intéresserons à une méthode plus complexe comme l'algorithme NEAT (Neuroevolution of augmenting topologies) qui est un algorithme génétique basé sur des réseaux de neurone, cela pourrait permettre d'ajouter un grand niveau de complexité et d'apprentissage à notre IA.

Enfin, nous pourrons mettre en place et comparer le respect des contraintes temps réels en fonction de l'algorithme utilisé.