Compte rendu de MPIA

Introduction

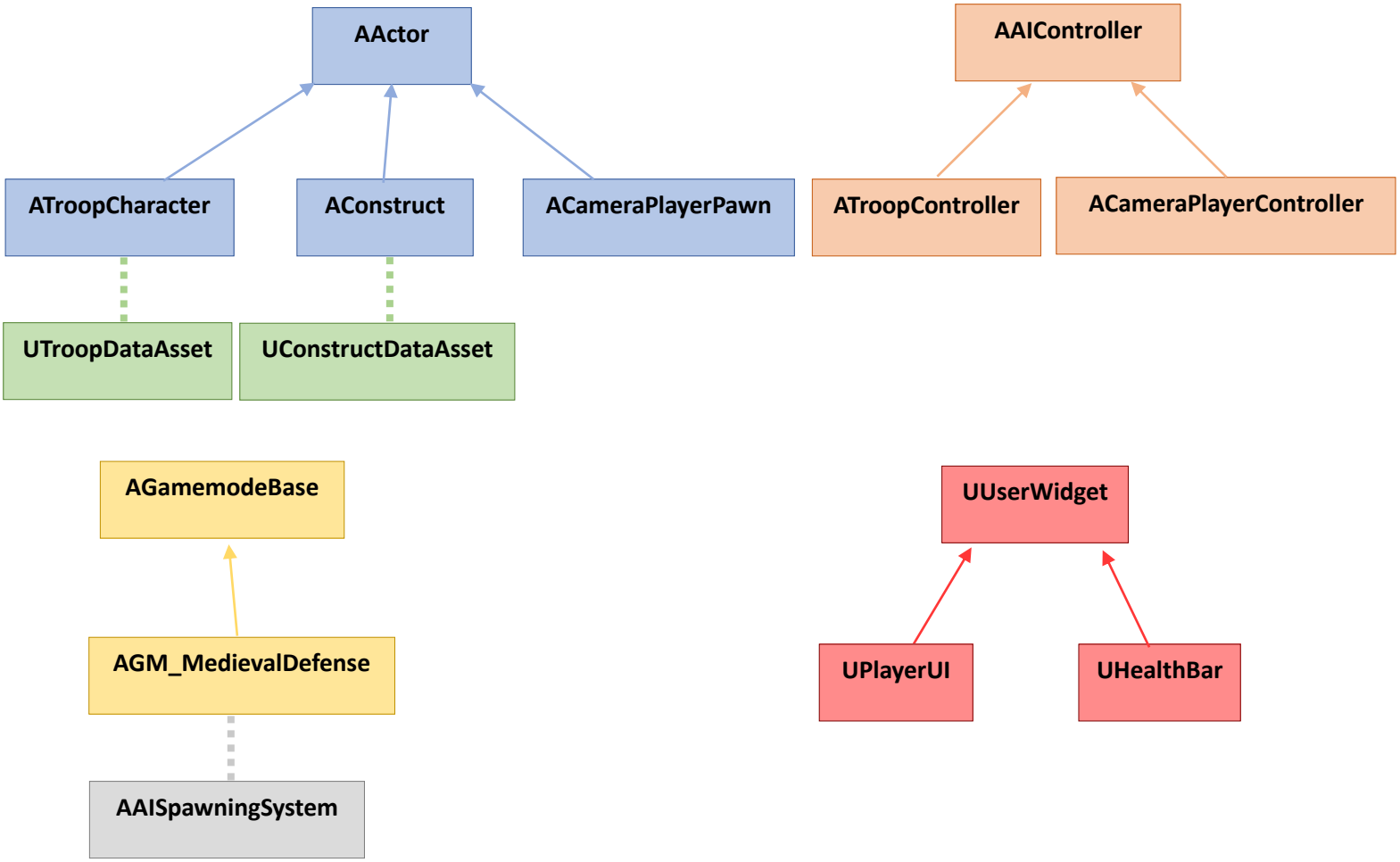
Ce compte rendu traite du projet réalisé durant les deux semestres du Master 2 ILJ de la faculté de Jean Perrin de Lens. Le sujet était la création d'un jeu de type « *Tower Defense* » sous Unreal Engine 5. L'idée initiale choisis a été un univers médiéval semi-réaliste ou le joueur doit défendre son château des vagues d'ennemis qui tentent de l'attaquer.

Nous verrons dans un premier temps l'architecture du projet, puis nous nous attarderons sur l'intelligence artificielle implémentée. Enfin, nous terminerons par présenter les contrôles et les bugs présents dans le jeu.

Compte rendu de MPIA

Sommaire

I/ Architecture du jeu



Les diagrammes UML ci-dessus ne représentent qu'une partie des classes crées pour la réalisation du jeu, il en existe en réalité bien d'autres. Cependant, elles suffisent à comprendre comment est agencé le projet.

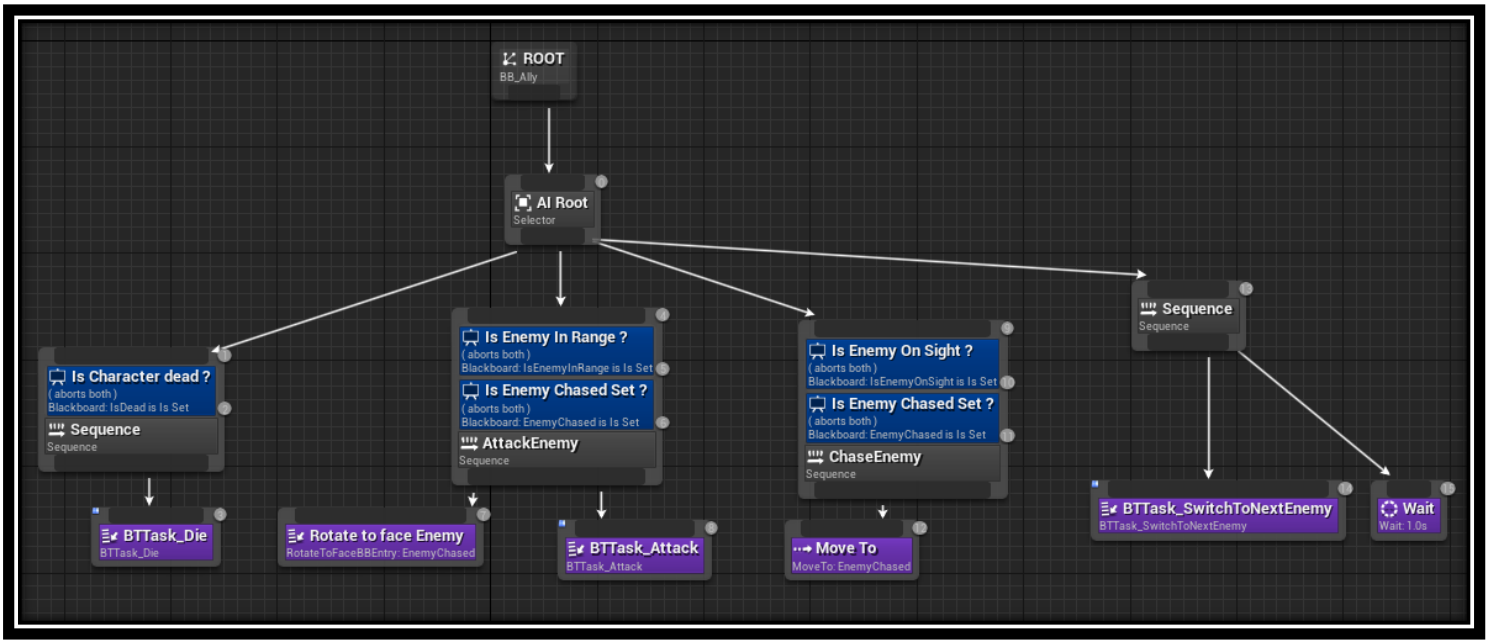
Classes	Description
ATroopCharacter	Classe de base des personnages du jeu. Elle contient tous les composants nécessaires à leur bon fonctionnement (comme *LifeComponent* qui gère la vie du character). Elle contient également un objet *UTroopDataAsset* qui permet de modifier facilement des valeurs, comme la vitesse de mouvement, le nombre de points de vie ou encore les points d'attaque. Ce data asset possède également un *GameplayTag* permettant de différencier les alliés des ennemis.
ATroopController	*Controller* lié à *ATroopCharacter*. Il se charge du déplacement du character, mais possède également les composants de perception (*AIPerception*) pour permettre aux unités de s'attaquer.
ACameraPlayerPawn	Classe que contrôle le joueur. Elle lui permet de se déplacer sur la carte pour qu'il puisse positionner ses troupes et gère les inputs liés à la souris. Une autre classe lui est associée pour restreindre ses mouvements dans une zone prédéfini.
ACameraPlayerController	*Controller* lié à *ACameraPlayerPawn*. Il permet de déplacer la caméra et contient également les interfaces utilisateurs utilisées pour invoquer les unités ou quitter la partie.
AConstruct	Classe de base des constructions du jeu. Cette classe n'est utilisée que pour le château, mais devait cependant se déclinée pour permettre au joueur de mettre en place des barricades, restreignant les déplacements ennemis. Par manque de temps cette fonctionnalité n'est pas présente dans le jeu. Cette classe comporte elle aussi un *LifeComponent*.
AGM_MedievalDefense	Classe représentant le mode de jeu. Elle découpe le jeu en rounds, eux même coupés en deux phases : une phase de 60 secondes où le joueur peut positionner ses troupes, et une phase d'assaut. Elle possède le système d'intelligence artificielle d'apparition des ennemis. Les groupes d'ennemis qui font le plus de dégâts seront plus nombreux à cet endroit au prochain assaut.
UPlayerUI	*Widget* du joueur comportant les boutons pour invoquer des troupes, le timer de la phase de préparation ainsi que le nombre de round.
UHealthBar	*Widget* représentant la barre de vie d'un acteur.

Certaines de ces classes *C++* est ensuite dérivée en *Blueprint* afin de créer, par exemple, les différents personnages, les différentes barres de vies, ect…

II/ Intelligence Artificielle

1 - Les arbres de décision

Pour définir le comportement des entités, il a été décidé d'utiliser les arbres de décision. Il y en a 2 : Le behavior tree des alliés et le behavior tree des ennemis. Le deuxième découlant du premier, commençons par voir celui des alliés.



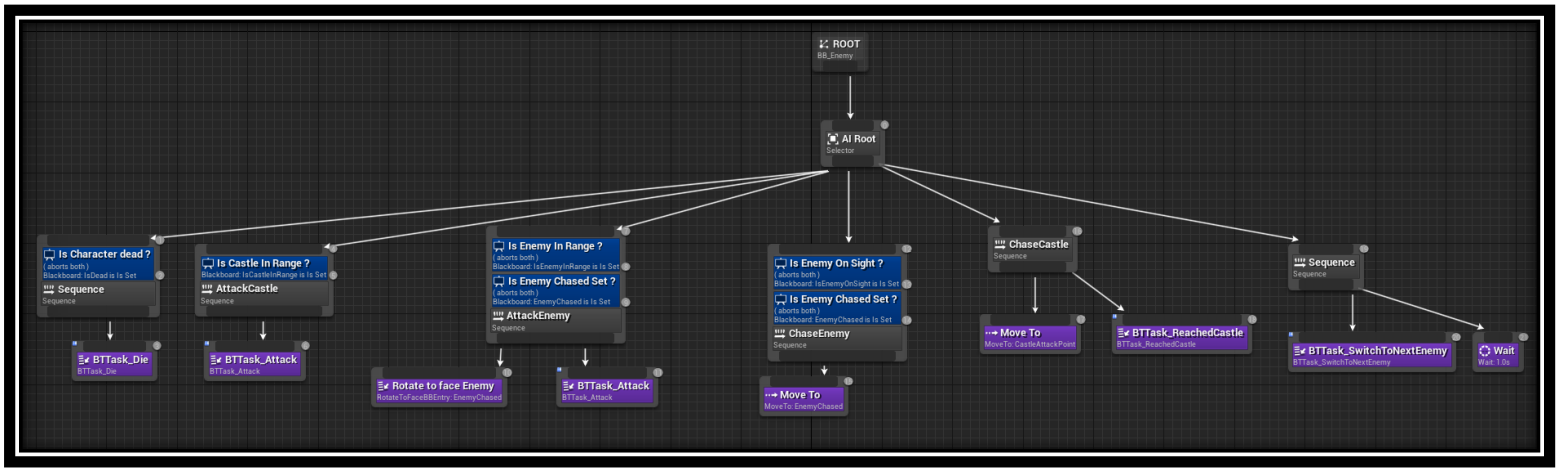
L'ordre de priorité des tâches varie de gauche à droite. Une tâche à gauche d'une autre sera exécutée en première.

Comme on peut le voir ci-dessus la tâche la plus important pour un allié est la mort. S'il n'a plus de point de vie il doit déclencher la tâche *Die* qui aura pour effet de jouer une animation et de le supprimer du monde.

La seconde tâche est l'attaque. Si une cible est désignée et qu'elle se situe dans la zone d'attaque de la troupe, alors la troupe se tourne vers elle et l'attaque.

La troisième tâche permet, si une cible est désignée et qu'elle se situe dans le champ de vision de l'entité, de se déplacer jusqu'à elle, dans le but de l'attaquer.

Enfin la dernière tâche est une séquence rajoutée pour éviter un bug. Nous en reparlerons en troisième partie.



Comme on peut le voir, l'arbre de décision des ennemis est très similaire à celui des alliés. Les différences interviennent au niveau de la deuxième et de la cinquième tâche.

La deuxième tâche a pour objectif de faire attaquer le château à l'ennemi si ce dernier se situe dans la zone attaquable du château.

La cinquième tâche, quant à elle, permet à un ennemi de se déplacer jusqu'à la zone attaquable du château.

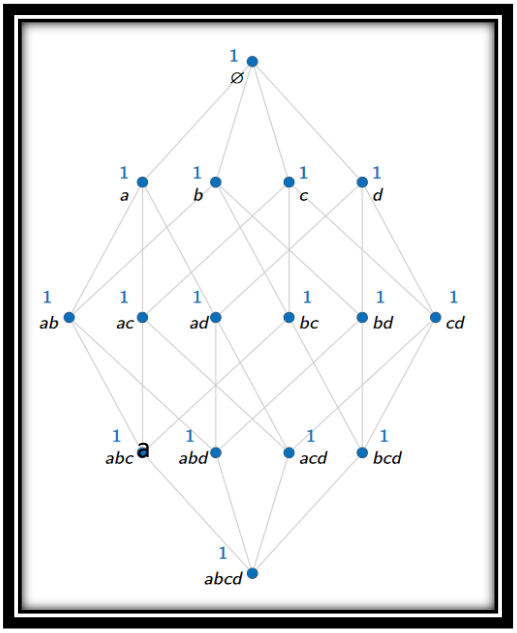
Ainsi, on comprend qu'un ennemi attaquera en priorité le château s'il le peut, puis les ennemis aux alentours. Mais si rien n'interfère l'ennemi se dirigera simplement vers le château. Enfin, on retrouve le dernier nœud de l'arbre pour corriger le bug dont nous parlerons plus tard.

II/ Intelligence Artificielle

2 – Le système de spawn

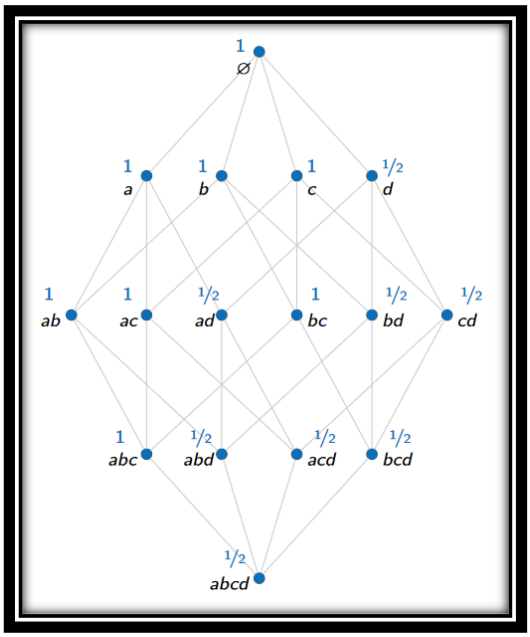
Le système d'apparition des ennemis reprend le principe de l'algorithme du « Weighted Majority » vu en cours de **PAA**. L'idée de cet algorithme est de donner un poids équivalent aux différentes possibilités que le système peut proposer, puis de les ajuster en fonction des réponses obtenus.

Compte rendu de MPIA



a	b	c	d	label
1	1	1	0	1

Adaptation des poids en fonction de la réponse obtenue



La classe *AGM_MedievalDefense* possède un acteur *AAISpawningSystem*. Ce dernier contient tous les acteurs de type *AEnemySpawner* (classe créée *C++* puis instanciée en *Blueprint* pour être positionnées dans le monde). Cet acteur commence par donner un *spawning rate* équivalent à chaque *spawner* (notre jeu possédant 3 *spawners*, chacun commence avec un taux d'apparition de $\frac{1}{3}$).

Chaque ennemi est lié à son *spawner*, si bien que lorsqu'il fait des dégâts, ces dégâts s'ajoutent à un total contenu dans le *spawner*.

À la fin de chaque round, *AAISpawningSystem* regarde quel *spawner* a récolté le plus de dégât par rapport au nombre d'ennemis qu'il a fait apparaître, puis ajuste les ratios d'apparition en conséquence. Les *spawners* ayant fait les moins bons scores se retrouvent à perdre un tier de leur capacité d'apparition (division par 1,5). Ces quantités perdues sont ensuite ajoutées au ratio du meilleur *spawner*.

On obtient donc un système d'apparition intelligent qui cherche les points faibles dans le jeu adverse pour optimiser ses dégâts.

III/ Contrôles et bugs du jeu

1 – Les contrôles

- **Clic gauche :** Le clic gauche sert à la sélection. Vous pouvez cliquer sur une de vos troupes pour la sélectionner, ou bien cliquer enfoncer pour sélectionner une zone. Les alliés sélectionnés apparaitront avec un contour vert. Pour désélectionner vos troupes, vous pouvez cliquer n'importe où ailleurs.
- **Clic droit :** Le clic droit sert à faire déplacer les troupes sélectionnées à la position cliquée.
- **Souris :** Pour déplacer la caméra dans le jeu, vous pouvez simplement positionner la souris sur les bords de l'écran vers lesquels vous souhaitez aller.
- **Espace :** Espace permet de pencher la caméra de bas en haut et inversement pour avoir un meilleur angle de vu.
- **Echap :** Echap ouvre le menu permettant de revenir à l'écran titre.

III/ Contrôles et bugs du jeu

2 – Les Bugs

Malgré mon travail certains bugs que je n'ai pas réussi à corriger persistent dans le jeu :

- **Bug de spawn :** Il arrive que des ennemis apparaissent en étant bloqués. Ils ne courront pas vers l'objectif et seront inatteignables. Il faut donc relancer la partie car la phase d'assaut ne peut pas être terminée.
- **Bug de perception :** Il arrive que des troupes adversaires passent l'une à coté de l'autre sans se détecter. Cela se produit lorsque, dans l'arbre de décision, la cible chassée est à *None* alors que le tableau de cible dans son *controller* n'est pas vide. C'est pour cela qu'on retrouve au niveau des arbres de décision un dernier nœud qui permet, si l'acteur n'a rien d'autre à faire, d'appeler la fonction *SwitchToNextEnemy* chaque seconde. Grâce à cela un acteur bugué se débuguera de lui-même.
- **Bug de zone d'attaque :** Les ennemis arrivant de face peuvent parfois attaquer le château au niveau de son côté gauche, ce qui ne devrait pas arriver.
- **Bug de sélection :** Il arrive que la sélection de zone ne commence pas là où l'on clique initialement.