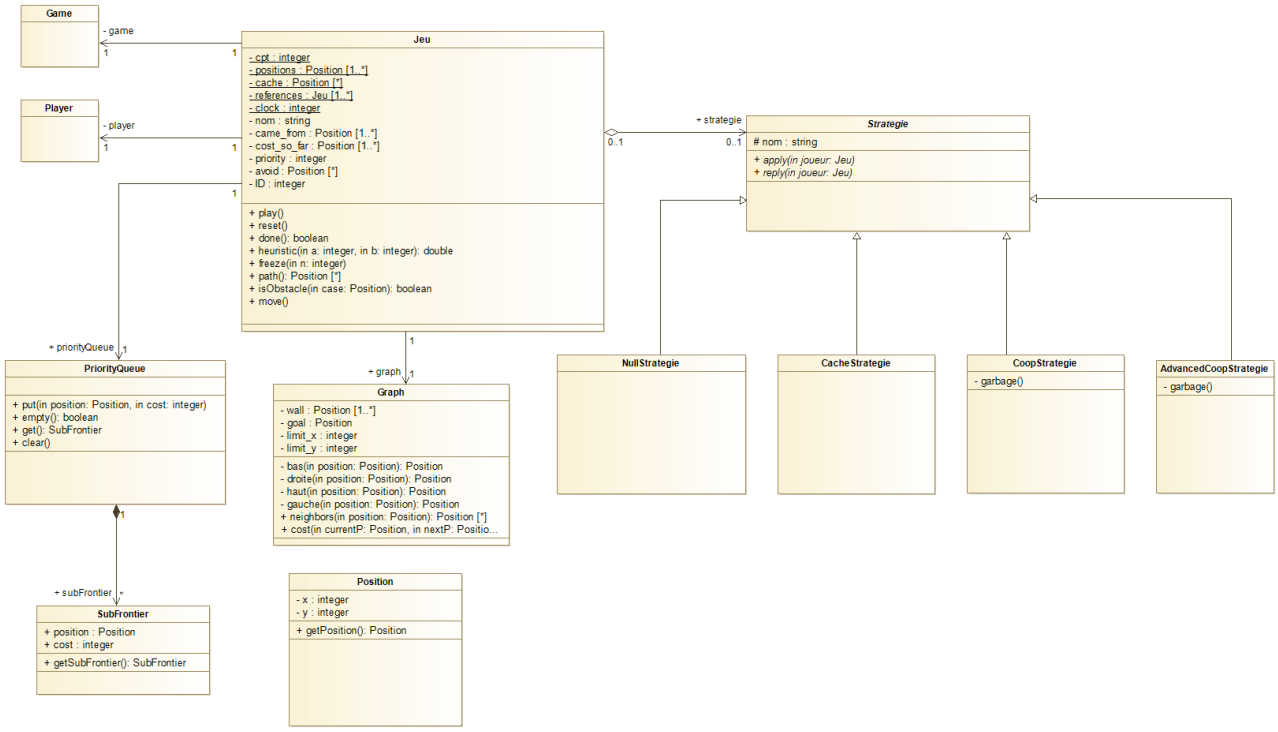
Ryan Ohouens **25/03/2019**

Projet A* CoopPathFinding

3I025 **Nicolas MAUDET**

I – Structure du projet



Ici une classe Jeu est utilisé pour représenter les différents agents.
Pour gérer la communication entre les différents agents des références comprenant tous les agents et leur position courante ont été utilisés. Les agents ont aussi une stratégie en commun à l'initialisation.

Le Design Pattern Stratégie à été utilisé pour modeler les différentes stratégies nécessaire afin d'éviter les collisions.
Cela permet une implémentation rapide et cohérente des stratégies et permet aussi de passer d'une stratégie à une autre au cours de l'exécution afin de répondre aux besoins des différents types de collisions possibles.

Le cœur du programme se situe dans les fonctions play() et move() de l'agent.
play() va trouver le chemin le plus court au goal attribué selon l'algorithme A*. Quand le chemin est défini la stratégie peut modifier son chemin grâce à l'appel apply().
Move() se contente de déplacer l'agent selon le chemin défini auparavant. Cependant si une collision est détecter avec un autre agent la stratégie peut modifier leur chemin grâce à l'appel de reply().
De ce fait la stratégie peut altérer le comportement des agents à deux moments cruciaux, avant qu'ils ne bougent et quand ils sont bloqués.

II – Les stratégies

Pour tester les différentes stratégies deux test ont été effectués. Le premier test est le test de terminaison, le deuxième est le test de rapidité.

Le protocole du test de terminaison est le suivant. Les agents doivent récupérer le plus de fioles possibles sur un grand nombre n d'itérations, prenons n=500. Pour cela il y autant de fioles que d'agents et à chaque fois qu'un agent ramasse une fiole une nouvelle est placé au hasard et il doit de nouveau la chercher.
Le test est raté si après une collision des agents sont dans l'incapacité de bouger et est réussi sinon.
Ce test si répété un grand nombre, permet aux agents d'emprunter une infinité de chemin et donc une infinité de cas de collisions afin de juger si la stratégie est efficace.

Le protocole du test de rapidité est le suivant. Les agents doivent récupérer la fiole qui leur est attribué tout en assurant que le groupe d'agent termine le plus vite possible. Pour cela il y autant de fiole que d''agents et dès qu'un agent à récupérer une fiole il ne doit plus bouger.
Ce test répété sur un grand nombre de cas, permet de comparer l'efficacité de la gestion du temps de collision entre les différentes stratégies.

Nous avons effectué ces test sur 5 stratégies :

NullStrategie :
Apply : Ne fait rien
Reply : Ne fait rien

CacheStrategie :
Apply : Ne fait rien
Reply : Revient sur ses pas si l'agent n'a pas la priorité
Ne fait rien si l'agent à la priorité
La priorité est défini par l'ordre d'initialisation des agents

OppoStrategie :
Apply : Ne fait rien
Reply : bloque l'agent ayant causé la collision pendant n tour, recalcul le chemin en considérant l'autre agent comme obstacle

CoopStrategie :

Apply : Si une partie du chemin de l'agent A est déjà prévu d'utilisation par l'agent B, l'agent A se stop pendant les n tours qu'il reste à l'agent B pour finir son chemin.
Reply : Ne fait rien

AdvancedCoopStrategie :

Apply : Si la partie du chemin Cy que l'agent A veut emprunter au temps t est déjà prévu d'utilisation par l'agent B au même temps, l'agent A se met en pause au temps t sur la partie du chemin Cy-1.
Reply : Ne fait rien

Résultat du test de terminaison sur 10 lancements :

1 : OppoStrategie (90%)
2 : CacheStrategie(50%)
3 : CoopStrategie, AdvancedCoopStrategie, NullStrategie (0%)

Les stratégies qui ne modifient pas leur chemin en situation de reply ont un taux de terminaison nul. Cela peut être du au fait que ces stratégies sont inefficaces en situation de jointure de début/fin de chemin.La jointure survient car ces stratégies ne modifient pas le chemin de A*, elle le retarde. La stratégie de cache peut être efficace dans une situation de jointure partiel, cependant en situation de jointure complète elle reste inefficace. La stratégie opportuniste est inefficace seulement si elle rentre dans un cycle d'attente. Ce qui est rare.

Résultat du test de rapidité sur 1 carte donnée sans situation de jointure :

1 : CacheStrategie, NullStrategie (17)
3 : AdvancedCoopStrategie (18)
4 : OppoStrategie (23)
5 : CoopStrategie (31)

Résultat du test de rapidité sur 1 carte avec situation de jointure partiel :

1 : OppoStrategie (25)
2 : CacheStrategie (34)
3 : CoopStrategie, AdvancedCoopStrategie, NullStrategie (null)

D'après les résultats la stratégie opportuniste est la plus stables selon les

situations. Cependant elle n'est pas forcément la stratégie la plus efficace.
La stratégie coopérative avancé pourrait gagné en stabilité si on pouvait lui permettre de modifier son A* lors de l'apply afin d'éviter les jointures.
Pour cela il faudrait modifier l'heuristic de manhattan appliqué a A*.
La stratégie coopérative est en général bien plus lente que les autres stratégies donc même avec plus de stabilité cela n'aurait pas d'importance sur notre objectif pour le groupe d'agents.
La stratégie de cache pourrait gagner en rapidité et en stabilité si on pouvait lui permettre de s'écarter de son chemin et celui de l'agent qui le bloque des que possible.

En conclusion aucune stratégie ne prédomine sur les autres, selon les cas une stratégie sera plus utile, c'est pour cela que le fait de pouvoir changer de stratégie en cour d'exécution du programme est important.
De plus stratégie une stratégie d'élection se basant sur l'attribution de la fiole à l'agent le plus proche réduirait considérablement le nombre de collisions.