Système multi-agents : Le monde des blocs

Hamdi Yacine p1709958 - Pont Louis p1704091

Lien vers le git : [TP_OVR - Github]

Ce document résume nos raisonnements et nos choix de conceptions pour le projet “Systèmes multi-agents : le monde des blocs”. Le but est de résoudre un problème de disposition de blocs. Chaque bloc possède un but, c’est-à-dire un bloc sur lequel il doit se trouver pour être satisfait. On suppose les choses suivantes :

- On dispose de 3 piles. En effet, le problème avec 2 piles est n’est pas résoluble et il est possible de résoudre tous les cas en se réduisant à 3 piles.
- Un bloc est représenté par un agent. Deux agents ne peuvent pas avoir le même but.
- Un agent ne peut percevoir que l’agent en dessous de lui, il peut pousser l’agent au-dessus et se déplacer vers une pile lorsqu’il est libre (pas de bloc au-dessus).

Version n°1

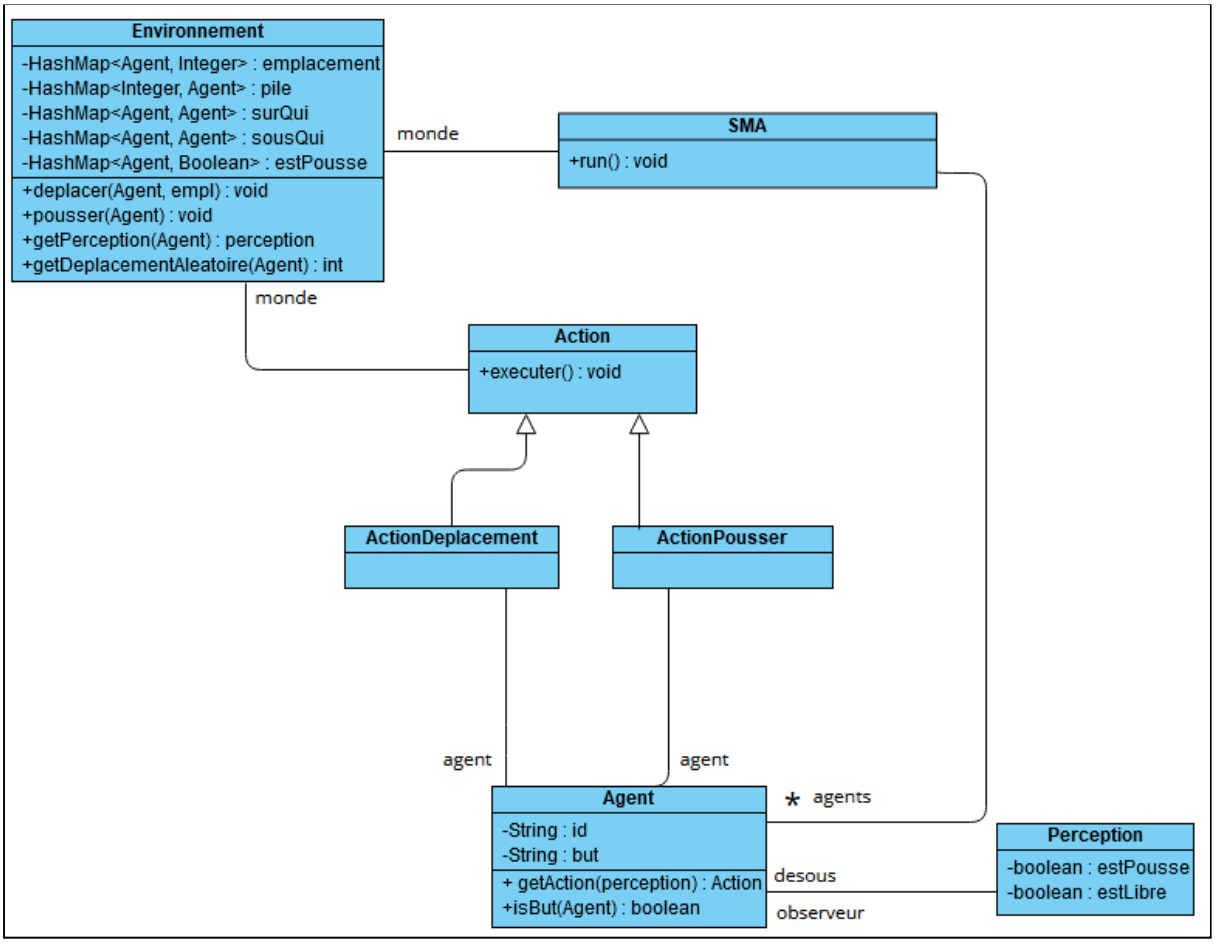

Diagramme de classe de la première partie

La principale problématique de ce sujet a été de définir l'environnement dans lequel évoluent les agents. Pour se rattacher le plus à la réalité, les agents n'ont pas d'accès direct à toutes les informations du monde, leur perception est réduite. Dans notre représentation du problème, nous avons décidé de coder l'environnement comme un ensemble de Map contenant les informations liées à chaque agent. Cette représentation est très utile pour accéder facilement et de manière performante à la perception de chaque agent, mais a pour contrainte de devoir modifier beaucoup de Map lors d'un simple déplacement d'agent.

- **emplacement** (Agent -> entier) : Le numéro de pile où se trouve l'agent donné
- **pile** (entier -> Agent) : L'agent qui se trouve à la tête de la pile donnée
- **surQui** (Agent -> Agent) : L'agent sur lequel se trouve l'agent donné
- **sousQui** (Agent -> Agent) : L'agent sous lequel se trouve l'agent donné
- **estPousse** (Agent -> Booléen) : Est-ce que agent donné est poussé ou non

Explication de la stratégie de résolution :

Tant que le problème n'est pas résolu, on parcourt la liste des agents qui prendront eux-même une décision en fonction de leur perception. En particulier, un agent se déplace aléatoirement s'il n'est pas satisfait ou s'il est poussé. S'il ne peut pas se déplacer, il pousse l'agent au-dessus.

Résultats :

On génère une liste d'agents que l'on dispose aléatoirement. On observe le nombre d'itérations et le temps d'exécution. Le tableau suivant montre la moyenne des valeurs calculées sur 4 dispositions différentes aléatoires

Nombre d'agents	Itérations	Temps (ms)
4	15	2.998
10	221	19.8825
20	1348	114.5

Version n°2

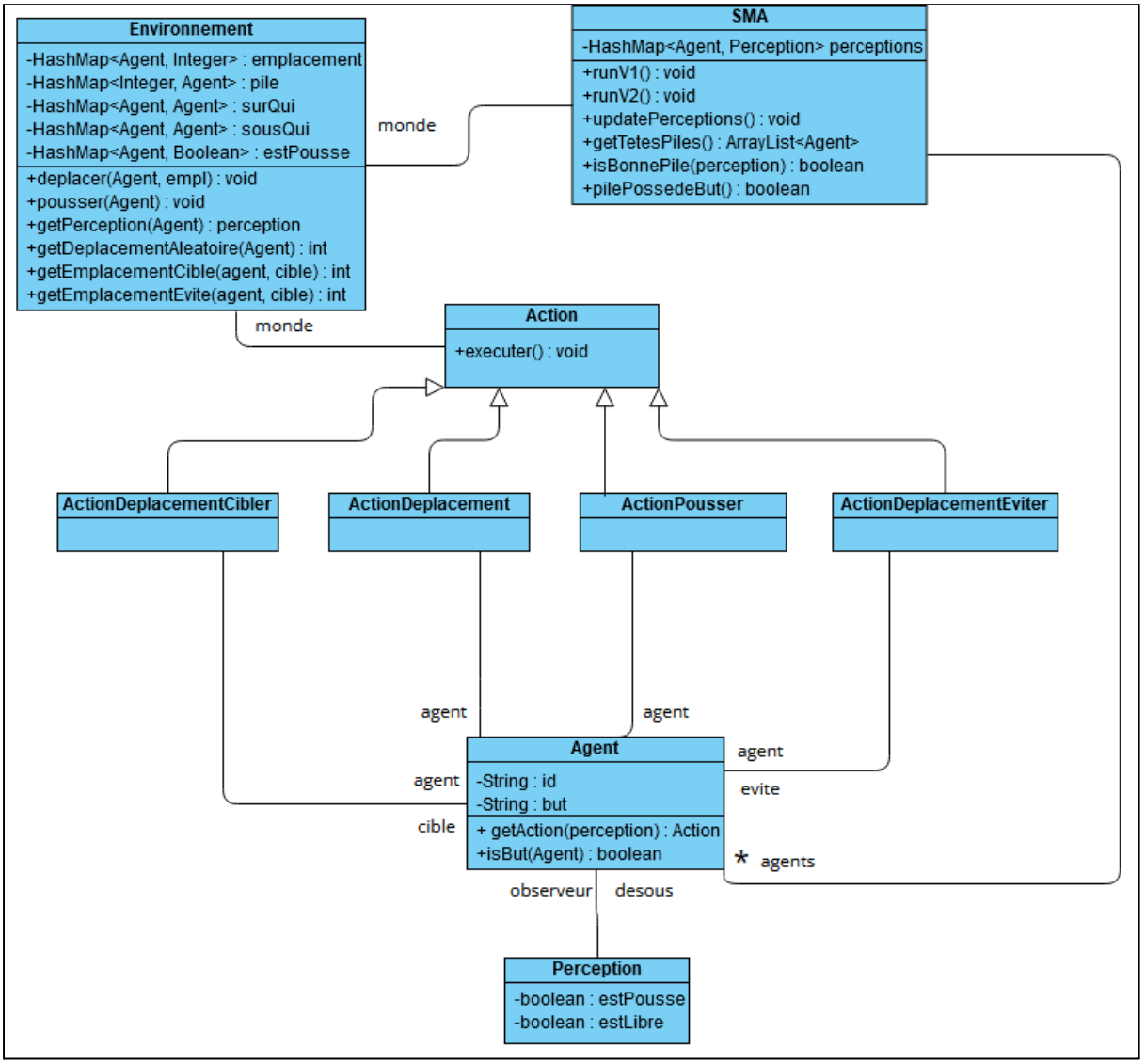

Diagramme de classe de la seconde partie

Principe : On se base sur la perception de tous les agents pour trouver la meilleure pièce à déplacer à chaque itération. On doit donc savoir quel agent en tête de pile il est le plus intéressant de déplacer et quelle stratégie de déplacement il faudra employer.

Hypothèses importantes :

- Comme un agent peut toujours connaître la pièce sur laquelle il est, alors il peut choisir d'éviter ou de cibler une pièce en tête de pile lorsqu'il se déplace.
- La classe "SMA" se base sur les perceptions locales de tous les agents pour planifier les décisions. Les agents ont moins d'impact individuel (pas de poussée). C'est la classe SMA qui détermine les actions à réaliser en analysant l'ensemble des perceptions. Sur le terrain, cela se concrétise par un “ordinateur” central qui contrôlera les agents lors de la résolution du problème.

La planification distingue deux cas principaux :
1) Il existe une pile quasi-satisfaite (toutes les pièces y sont bien placées).

- S’il n’y a qu’une seule pile, alors le problème est résolu.
- Si la pièce qui peut continuer la pile quasi-satisfaite est libre, alors elle se déplace en la ciblant.
- Sinon, on déplace la pièce gênante en évitant la pile quasi-satisfaite.

2) Il y n'a pas de pile quasi-satisfaite.

- S'il n'y a qu'une seule pile, on déplace l’agent aléatoirement.
- Deux piles : On essaye de déplacer la pièce dont le but est le sol (racine). Sinon, on déplace la pièce qui gêne.
- Trois piles : On essaye de fusionner les deux piles qui ne possèdent pas la racine. Ceci permet de “créer de la place” pour la racine.

Remarque : Toutes les opérations de la fonction se basent exclusivement sur les perceptions des agents. Pour connaître les têtes de piles, on regarde les agents qui sont libres en parcourant la liste des perceptions. Pour savoir si une pile est quasi-satisfaite ou si elle possède une certaine pièce, on le détermine par récursivité sur les perceptions (voir les fonctions *isBonnePile* ou *pilePossedeBut*).

Résultats :

Nombre d’agents	Itérations	Temps (ms)
4	7	1.69
10	28	3.83
20	95	36.3

On remarque que cette stratégie permet une planification beaucoup plus performante en temps et en nombre d’itérations.

Exécuter le programme ;

1) Lancer le fichier “launcher.bat”
   ou
   Lancer la commande suivante dans une console : `java -jar "TP_OVR.jar"`

2) Entrer le nombre d’agents à générer.

3) Sélectionner la version à exécuter (1 ou 2).

4) Le déroulement du programme s’affiche : état de l’environnement avant chaque itération et description de l’action effectuée.