


22/12/2020

Interaction Multi-Agents

Problème du puzzle



Marie Jassigneux
Pierre-Henri Chupin

Table des matières

Choix du langage	2
Choix de conception	2
Agent	2
Environnement.....	3
Index2D	3
Message.....	3
MessageBox.....	3
Conclusion	4

Choix du langage

Ce TP a été réalisé en Java car c'est un des langages qu'on maîtrise le plus et qu'il semblait être un des plus appropriés pour ce TP. On utilise les librairies JavaFX pour tout ce qui concerne l'affichage de la grille.

Choix de conception

Nous sommes partis sur une interaction perception/action. Pour cela nous avons conçu notre projet autour de nos agents et de l'environnement. Nous avons aussi créé d'autres classes Java qui nous semblaient essentielles tel que la gestion de la communication entre nos différents agents.

Agent

Tout d'abord, il faut savoir que nos agents agissent de façon dynamique et tous en même temps. En effet, nous utilisons du multi-threading pour ce TP.

Notre classe agent est composée de différents paramètres :

- finalPosition : contient les coordonnées finales de notre agent. C'est-à-dire l'objectif de ce dernier.
- Id : l'id de l'agent actuel.
- currentPosition : contient les coordonnées de la position actuelle de l'agent.
- Environnement : ce qui nous permet de faire le lien entre notre agent et l'environnement.

Notre agent effectue la même suite d'actions en boucle tant que l'environnement n'a pas fini.

- Tout d'abord, il regarde sa boîte de message.
- Si elle n'est pas vide alors il va d'abord la nettoyer avant de pouvoir regarder le premier message.
 - Si ce message est une demande pour qu'il bouge alors il va essayer de bouger et envoyer un message de réussite ou non à l'agent qui lui a envoyé cette requête.
 - Si ce message est un accusé de réception qui lui dit que la demande a été rejetée alors il ne fait rien en conséquence pour le moment. (C'est quelque chose que l'on pourrait développer plus tard : conflit entre différents agents)
- Si sa boîte de message est vide et qu'il n'est pas sur sa position finale alors il regarde la direction qui le rapproche le plus de sa position finale
 - S'il y a plusieurs positions qui ont une distance égale de sa position finale et qu'elles sont vides alors il choisit une de ses positions de manière aléatoire.
 - Si la position qui le rapproche le plus de sa position finale est un agent alors il envoie un message à l'agent positionné dessus pour qu'il se déplace.
- Puis il recommence ces actions.

Environnement

Notre classe Environnement est composé des paramètres suivants :

- grille : correspond à la grille de notre puzzle. Chaque case est du type agent.
- agents : correspond à la liste de nos agents.
- height, width : correspond à la taille de notre grille.
- messageBoxes : correspond à un tableau qui nous permet de stocker pour chaque agent sa boîte de message.
- nbAgent : correspond au nombre d'agent que nous avons dans notre environnement.

Tout d'abord c'est notre environnement qu'initialise le jeu et qui donne des positions aléatoires à chaque agent ainsi que leur position finale. On retrouve toutes les actions possibles de notre agent : move, nettoieBox, envoi (envoyer un message), setMessageBoxes.

On retrouve aussi des fonctions pour retrouver les positions des agents ou celle de ses voisins, ou de ses cases voisines. C'est aussi dans l'environnement qu'on lance nos différents threads.

Index2D

Cette classe est une structure que l'on a créée pour faciliter notre grille. On donne à chaque case [i,j] une valeur 2D ce qui permet de simplifier le code et sa compréhension.

Message

Cette classe est celle qui nous permet de définir un message. On a différents paramètres qu'on pourrait encore améliorer. Mais nous n'avons pas eu le temps de mettre en place la partie de réflexion de l'agent en fonction du type de réponse.

Notre classe Message est composé des paramètres suivants :

- receveur : correspond à l'agent qui va recevoir le message.
- envoyeur : correspond à l'agent qui envoie le message.
- reponse : un entier qui représente la réponse de la demande : 0 = OK, -1 = Je ne peux pas.
- whereR : correspond à la position du receveur
- whereE : correspond à la position de l'envoyeur.
- type : un entier qui correspond au type de message : 0 = accusé de réception, 1 = demande d'action à faire.

MessageBox

Cette classe correspond à la boîte à message des agents. Elle est composée d'une liste de message. Elle nous permet ainsi de gérer plus facilement les messages reçus par l'agent.

Conclusion

Pour conclure ce TP, nos agents agissent de façon dynamique et arrivent à leurs positions finales. Il reste de nombreux points qui n'ont pas été totalement finalisé et qu'on pourrait éventuellement reprendre plus tard. Notamment pour tout ce qui concerne la négociation.