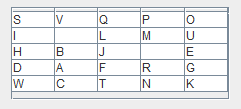
SMA - TP1

Interactions Multi-Agents

# Introduction

Dans le cadre de ce TP nous avons réalisé un puzzle qui se résout grâce à une approche multi-agents. Au sein d’une grille n fois n, les agents sont sous la forme de cases du puzzle. Chaque case apparait à une position aléatoire et a pour objectif d’atteindre une position de destination. Ainsi, en permettant aux agents de communiquer entre eux et de prendre des décisions non déterministes en tenant compte de l'environnement, le puzzle se résout progressivement.

Nous avons réalisé ce projet en java. Une simple interface graphique contient un tableau représentant la grille du puzzle, les agents sont identifiés par des lettres.

*Capture de notre interface en 5x5 avec 23 agents*

# Implémentation

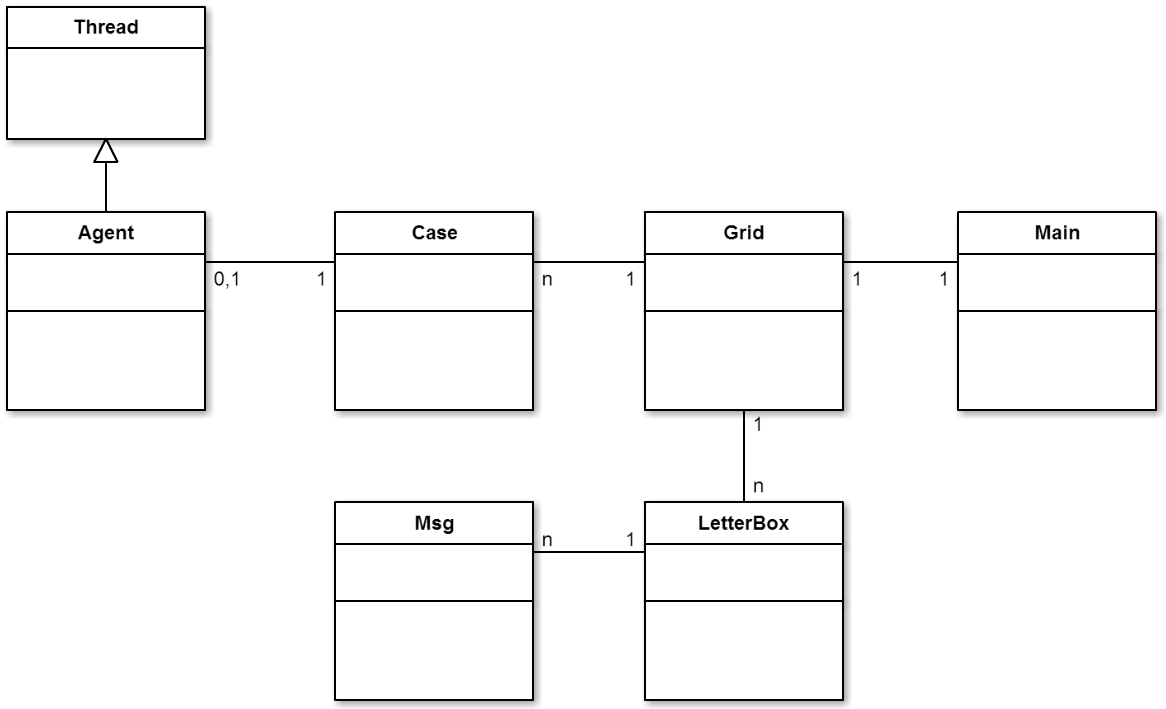
## Agents et messages

Chaque agent est modélisé par un Threads, exécutant une boucle à une certaine vitesse. Les agent communiquent entre eux grâce à une boîte au lettre. Celle-ci est constituée d’un singleton contenant une HashMap reliant une boîte privée à chaque agent. La boîte privée est une liste de messages non lus à destination du propriétaire de la boîte. Un message a la structure suivante :

{ émetteur, destinataire, requête, action, informations }. Cette structure représente le langage utilisé par les agents pour communiquer. Lorsqu’un agent consulte sa boîte au lettre, celle-ci est alors vidée. Nous avons entouré l’écriture et la lecture des boîtes avec des instructions synchronized pour éviter les problèmes de concurrence.

Plus concrètement, dans notre cas les messages vont avoir comme seule requête possible ASK, et comme seule action possible MOVE, ce qui signifie pour le destinataire une demande de la part de son voisin de s’écarter pour le laisser passer.

Ci-dessous, le diagramme de classes correspondant à notre implémentation.



## Mise à jour de l’agent

Avant toute chose, l’agent va consulter sa boîte au lettre pour vérifier si un autre agent lui demande de bouger.

Si l’agent n’est pas à sa place définitive ou si on lui demande de bouger, il va choisir une nouvelle case où se positionner. Dans le cas où aucune n’est libre autour de lui, alors il choisit la case C qui l’intéresse le plus et demande à l’agent dessus de bouger ; sauf dans le cas où l’agent sur la case C est justement celui qui vient de lui demander de bouger, auquel cas il choisit la 2ème case qui l’intéresse le plus.

Si des cases libres aux environs de l’agent le rapprochent de son objectif, il va alors en choisir une aléatoirement.

Sinon, il va soit se déplacer aléatoirement vers une case vide, soit envoyer un message a un voisin situé sur son chemin. Cette dernière décision étant aussi prise de manière aléatoire.

Si aucune case autour de lui n’est libre et qu’il n’est pas à sa place définitive ou qu’on lui a demandé de bouger, il va alors forcément demander à un voisin de bouger.

## Analyse des résultats

Pour une grille 5 par 5 et trois cases vides (ou plus), le puzzle est toujours résolu en quelques secondes. La communication entre les agents leur permet de gagner rapidement leur objectif tout en tenant compte des requêtes de leurs voisins.

Les résultats sont donc très bons, et les limites apparaissent quand on ne laisse qu’une ou deux cases vides.

## Limites

Le système fonctionne correctement dans la plupart des cas. Néanmoins, avec deux cases vides c’est généralement très long, et avec une seule case vide, on ne voit le puzzle résolu que très rarement. Quand il n’y a stagnation, c’est que le puzzle est rentré dans une phase où quasiment tous les blocs sont bien positionnés, et que un (ou un petit nombre) de blocs n’arrivent pas à atteindre leur destination facilement, c’est à dire sans que plusieurs blocs déjà bien placés ne se bougent.

Cependant, le programme des agents est développé sous forme d’un algorithme “de Las Vegas”. Ce qui signifie qu’il y a de l’aléatoire dedans, et que la convergence est assurée, bien que le temps ne le soit pas.

Donc même avec une seule case vide, notre système assure une convergence vers le bon résultat, sans pour autant garantir le temps d’exécution.

## Conclusion

La difficulté de ce TP résidait dans la gestion des objectifs des agents. En effet, chaque agent avait un objectif individuel : celui de rejoindre sa case finale ; mais ils avaient aussi un objectif collectif : celui de finir le puzzle. Le point épineux de l’implémentation était donc d’être capable de gérer ces deux objectifs simultanément, que nous avons traité grâce à la communication des agents, qui permet pour un agent donné de prendre en compte les objectifs de ses voisins.