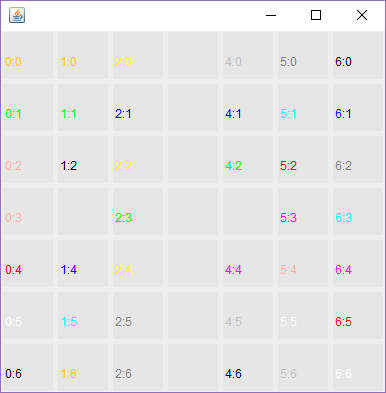
SMA - Cognitif

TP1 : Taquin



2015

Adrien Castex – Benoit Vuillemin

POLYTECH

18/11/2015

Table des matières

[Introduction 2](#_Toc436759229)

[Algorithmes de déplacement 2](#_Toc436759230)

[Optimisations 2](#_Toc436759231)

[Historique des mouvements 2](#_Toc436759232)

[L’algorithme de plus court chemin est stochastique 2](#_Toc436759233)

[Démarrage 2](#_Toc436759234)

[Traitement d’un message « move » 2](#_Toc436759235)

[Instabilité 2](#_Toc436759236)

[Informations stockées dans les messages 2](#_Toc436759237)

[Positionnement de départ 3](#_Toc436759238)

[Mesures 4](#_Toc436759239)

[Conclusion 5](#_Toc436759240)

# Introduction

Ce TP de SMA reprend le jeu classique du taquin. Cependant, chaque bloc se déplace en suivant son propre algorithme. Pour réussir à atteindre la case recherchée, un système de communication a été établi entre les blocs. Ceci permet à un bloc qui bloque le passage de se déplacer afin de laisser passer le bloc qui est coincé. Ce rapport permet de détailler le fonctionnement de ce projet.

# Algorithmes de déplacement

*AStar*: Cet algorithme calcule la distance entre deux cases.

*DirectPath*: L’agent se déplace de façon directe sans chercher de chemin alternatif qui pourrait être plus court (chemin libre).

# Optimisations

## Historique des mouvements

Si l’agent a le choix entre plusieurs déplacements, alors il va choisir celui qu’il a utilisé le moins récemment.

## L’algorithme de plus court chemin est stochastique

On trie les différents chemins de façon aléatoire avant de les ordonner en fonction de leur distance avec la destination. Cela permet de varier le chemin si l’un a l’air plus bloqué qu’un autre chemin équivalent, sans avoir besoin de prendre en compte qu’un chemin peut être plus long à se déboucher qu’un autre.

# Démarrage

On démarre les agents à des moments différents pour leur permettre d’agir à des moments différents les uns des autres. Pour faire cela, au moment où l’agent est lancé, nous le mettons en suspens (sleep) durant un temps aléatoire compris entre 500ms et 1500ms.

# Traitement d’un message « move »

Lorsque l’agent récupère un message « move » et qu’il ne peut pas se déplacer (aucune case disponible autour de lui), alors il envoie à tous ses voisins, sauf l’émetteur, une demande de « move ». Il remet ensuite le message qu’il vient de retirer dans sa boite aux lettres pour ne pas oublier qu’il n’a pas encore effectué son opération de « move ».

# Instabilité

Nous avons ajouté un paramètre d’instabilité des agents qui permet aux agents, de façon probabiliste, de sortir de leur case objective sans avoir reçu de demande. Cela permet, dans des situations encombrées, de débloquer plus rapidement la situation. Néanmoins, une trop forte valeur de ce paramètre peut aussi rendre le plateau plus encombré. En effet, un agent qui quitte sa position finale peut se retrouver sur le chemin d’un agent qui ne faisait que passer.

# Informations stockées dans les messages

* La position à partir de laquelle l’agent doit bouger
* La position de l’émetteur

Cela permet de savoir si le message envoyé est obsolète ou non.

On considère le message obsolète si la position à partir de laquelle on doit bouger n’est pas celle de l’agent qui reçoit le message et la position de l’émetteur n’est pas celle de celui qui a envoyé le message.

Si le message est obsolète, alors il est ignoré, sinon il est traité immédiatement.

# Positionnement de départ

Nous commençons par positionner les agents aléatoirement sur la grille.

Après avoir lancé les agents, ceux-ci se déplacent en toute autonomie vers leur objectif tout en traitant autant que possible les messages qu’ils reçoivent des autres agents.

Lorsque tous les agents sont totalement satisfaits (c’est-à-dire que tous les agents se trouvent à leur position finale) alors ils s’arrêtent (ils quittent leur thread).

# Mesures

|  |  |  |  |
| --- | --- | --- | --- |
| **Thread sleep (ms)** | **Instabilité** | **Couverture** | **Temps** |
| [50 ; 150[ | 0% | 26.5% | 2.4s |
| [50 ; 150[ | 0% | 26.5% | 2.8s |
| [50 ; 150[ | 0% | 26.5% | 2.4s |
| [50 ; 150[ | 0% | 26.5% | 2.8s |
| [50 ; 150[ | 0% | 26.5% | 2.4s |
| [50 ; 150[ | 0% | 26.5% | 2.8s |
| [50 ; 150[ | 0% | 26.5% | 2.4s |
| [50 ; 150[ | 0% | 26.5% | 2.8s |
|  | | **Moy :** | 2.6s |
| [50 ; 150[ | 5% | 51% | 4.8s |
| [50 ; 150[ | 5% | 51% | 10.8s |
| [50 ; 150[ | 5% | 51% | 6.4s |
| [50 ; 150[ | 5% | 51% | 6.8s |
| [50 ; 150[ | 5% | 51% | 3.6s |
| [50 ; 150[ | 5% | 51% | 14.8s |
| [50 ; 150[ | 5% | 51% | 19.6s |
| [50 ; 150[ | 5% | 51% | 5.6s |
|  | | **Moy :** | 9s |
| [50 ; 150[ | 0% | 65.3% | 6.8s |
| [50 ; 150[ | 0% | 65.3% | 46.5s |
| [50 ; 150[ | 0% | 65.3% | 8.8s |
| [50 ; 150[ | 0% | 65.3% | 8.8s |
| [50 ; 150[ | 0% | 65.3% | 8.4s |
| [50 ; 150[ | 0% | 65.3% | 13.2s |
| [50 ; 150[ | 0% | 65.3% | 6.9s |
| [50 ; 150[ | 0% | 65.3% | 7.4s |
|  | | **Moy :** | 13.35s |
| [50 ; 150[ | 0% | 73.5% | 23s |
| [50 ; 150[ | 0% | 73.5% | 20s |
| [50 ; 150[ | 0% | 73.5% | 12.4s |
| [50 ; 150[ | 0% | 73.5% | 18.4s |
| [50 ; 150[ | 0% | 73.5% | 19.2s |
| [50 ; 150[ | 0% | 73.5% | 15s |
| [50 ; 150[ | 0% | 73.5% | 25s |
| [50 ; 150[ | 0% | 73.5% | 15.6s |
|  | | **Moy :** | 18.575s |
| [50 ; 150[ | 0% | 81.6% | 271.8s |
| [50 ; 150[ | 0% | 81.6% | 227.3s |
| [50 ; 150[ | 0% | 81.6% | 133.3s |
| [50 ; 150[ | 0% | 81.6% | 348.2s |
| [50 ; 150[ | 0% | 81.6% | 387.5s |
| [50 ; 150[ | 0% | 81.6% | 355.1s |
| [50 ; 150[ | 0% | 81.6% | 30.2s |
| [50 ; 150[ | 0% | 81.6% | 117.3s |
|  | | **Moy :** | 233.8s |

En passant d’une couverture de 73.5% à 81.6%, nous avons multiplié par 12.6 le temps moyen de résolution.

# Conclusion

Ainsi, nous avons pu mettre en place des blocs qui peuvent interagir entre eux via un système de messagerie. L’algorithme est plutôt efficace si la couverture est inférieure à 70 %, comme on peut le voir sur les mesures.