Rapport SMA

Systèmes multi-agents

Antoine Ganne & Alicia Rocchia

01.2020

INTRODUCTION

Dans le cadre du projet de SMA, nous avons modélisé une solution a un problème de résolution de puzzle de type taquin.

C'est à dire que, sur une grille, différents agents cherchent à se positionner à un emplacement particulier de la grille. Cet emplacement étant unique pour chaque agent.

Nous avons décidé de coder ce projet sur Unity (et donc en C#) d'abord pour des raisons personnelles de gain d'expérience sur ce langage et parce que Unity permet de gerer relativement facilement les problèmes de multi-threading et de représentation visuelle. De plus, Unity permet de modifier, en cours d'exécution du programme, les attributs des agents ce qui aide beaucoup pour tester le comportement du code.

Nous avons mis en place un système de génération aléatoire du puzzle pour assurer que tout puzzle que nous générons admet une solution. En effet dans le cas du taquin(une seule case de libre) la moitié des configuration n'est pas résoluble. A l'instar du sudoku, nous mélangeons les agents à partir de leurs positions désirée.

Représentation visuelle

Les agents sont représentés par des pièces de jeu d'échec.

Lorsqu'un agent est placé (ou se déplace) sur sa case désirée alors cette dernière s'affiche en vert.

MODELISATION

Nous sommes partis sur une approche centrée sur les agents.

En effet l'environnement est très simple et se résume à une classe "manager" qui initialise le problème et une classe "board" qui met à jour l'état de la grille en temps réel. Tandis que les agents sont relativement complexes.

Un système d'échanges de messages entre agents a été mis en place (négociation décentralisée). Nous avons gardé deux types de messages :

- L'ordre de déplacement (lorsqu'un agent demande à un autre agent de se

déplacer). Les informations que ce message contient est l'émetteur, destinataire et le type de message.

- La réponse à l'ordre de déplacement (lorsqu'un agent signale aux agents qui ont demandé son déplacement qu'il s'est effectivement déplacé). De même que précédemment, Les informations que ce message contient est l'émetteur, destinataire et le type de message.

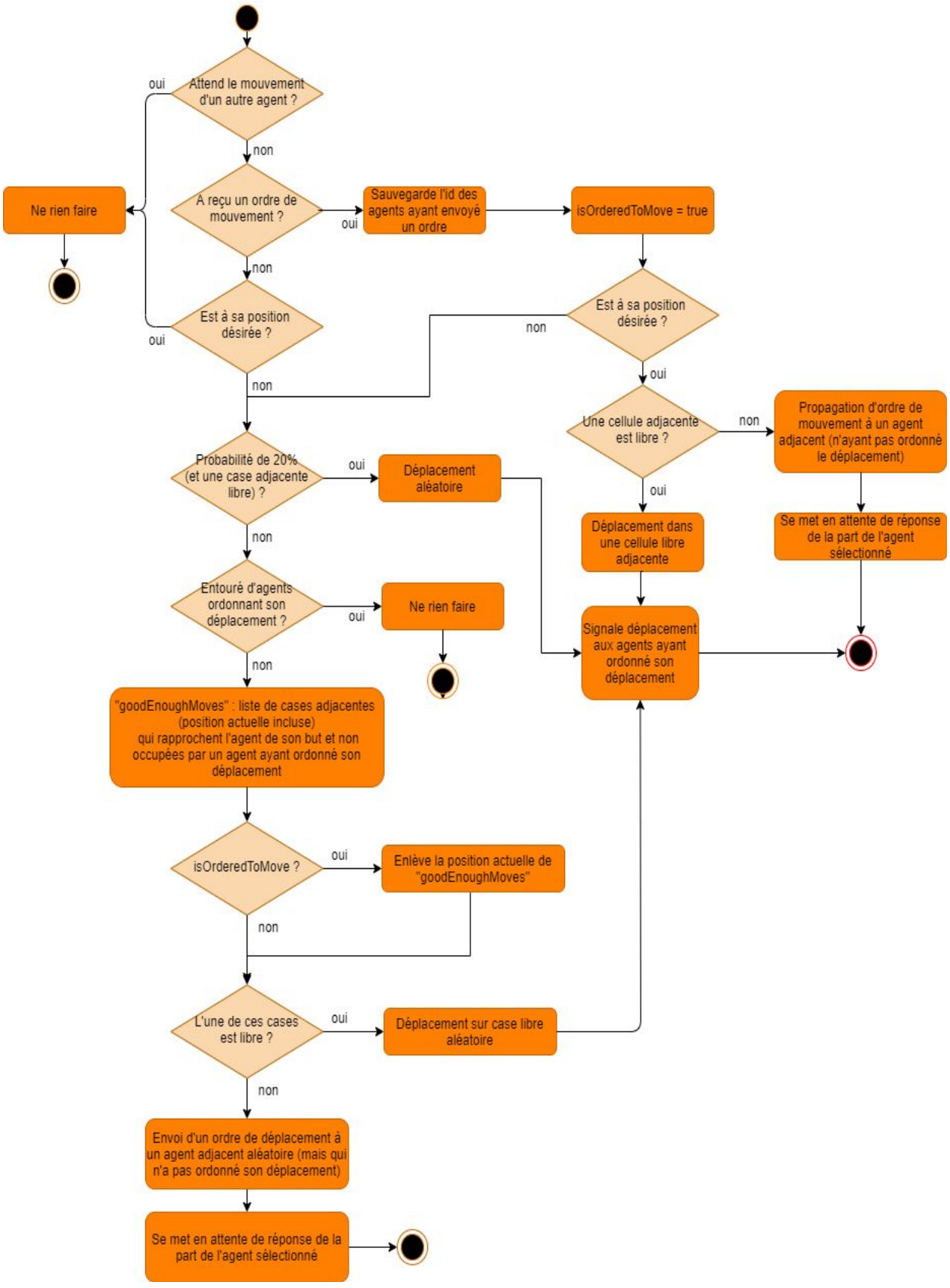
La partie centrale de notre projet est le processus de choix d'action de chaque agent. Nous nous sommes basés sur l'idée de faire des chaînes de propagation de message, c'est à dire que chaque agent cherche à se rapprocher de son but et, lorsqu'il est bloqué, cherche à former une chaîne.

Par exemple dans un cas où l'encombrement est très important, l'agent envoie un ordre de déplacement à l'agent qui le bloque et attend. Ce dernier va alors propager l'ordre de déplacement jusqu'à ce qu'un agent de la chaîne trouve une case vide où se déplacer. Le dernier agent de la chaîne va donc se déplacer et signaler son déplacement aux autres agents de la chaîne (par propagation). Ainsi les agents de la chaîne vont se suivre les uns après les autres.

Dans le meilleur cas, chaque agent de la chaîne s'est alors rapproché de son but.

Nous avons introduit de l'aléatoire pour débloquer les situations où les agents répètent les mêmes séquences d'actions. Cet aléatoire se manifeste par la sélection aléatoire des actions parmi les actions correctes (qui n' éloignent pas l'agent de son but) et par l'ajout de déplacement dans une case vide aléatoire dans 20% des cas.

Nous avons abouti au processus de choix d'action suivant :


Attend le mouvement d'un autre agent ?
oui
non
Ne rien faire
A reçu un ordre de mouvement ?
oui
Sauvegarde l'id des agents ayant envoyé un ordre
isOrderedToMove = true
non
Est à sa position désirée ?
oui
Est à sa position désirée ?
non
oui
non
Une cellule adjacente est libre ?
non
Propagation d'ordre de mouvement à un agent adjacent (n'ayant pas ordonné le déplacement)
Probabilité de 20% (et une case adjacente libre) ?
oui
Déplacement aléatoire
oui
Déplacement dans une cellule libre adjacente
Se met en attente de réponse de la part de l'agent sélectionné
non
Entouré d'agents ordonnant son déplacement ?
oui
Ne rien faire
Signale déplacement aux agents ayant ordonné son déplacement
non
"goodEnoughMoves" : liste de cases adjacentes (position actuelle incluse) qui rapprochent l'agent de son but et non occupées par un agent ayant ordonné son déplacement
isOrderedToMove ?
oui
Enlève la position actuelle de "goodEnoughMoves"
non
L'une de ces cases est libre ?
oui
Déplacement sur case libre aléatoire
non
Envoi d'un ordre de déplacement à un agent adjacent aléatoire (mais qui n'a pas ordonné son déplacement)
Se met en attente de réponse de la part de l'agent sélectionné

Resultats

Les résultats que nous obtenons sont très satisfaisants. En effet, nous arrivons à résoudre le problème sur une grille 3x3 même avec 8 agents (et donc une seule case de libre). Vous pouvez voir des exécutions du projet sur des grilles de différentes tailles sur les vidéos que nous vous avons fournis avec ce rapport. Le nom des videos correspond à la taille de la grille (3x3 ou 4x4) puis le taux de remplissage (ou “taquin” si une seule case de libre).

La vidéo “3x3_taquin_chaine” montre particulièrement la mécanique de chaîne, en plus de montrer la résolution d’un taquin.

On observe aussi que notre système est capable de résoudre des blocages relativement complexes dans la vidéo “4x4_75_bloquage_resolu”.

La vitesse de convergence sur les grandes grilles diminue car le puzzle peut prendre un temps plus important à être résolu.

Notre système de chaîne d’agents peut montrer ses limites sur des grandes grilles car :

- Plusieurs chaînes peuvent se former simultanément et se gêner
- Une chaîne peut former un cycle (boucler sur un agent déjà présent dans la chaîne)

Il peut aussi y avoir des situations bloquantes qui subsistent, notamment lorsqu’il ne reste qu’un seul agent à placer. Cette situation ne se produit pas lorsqu’il n’y a qu’une seule case libre.

Améliorations proposées

Nous pourrions mettre en place une limite au nombre d’ordres envoyés d’un agent à l’autre afin de sortir de certaines situations bloquantes.

Aussi, le système de chaîne pourrait être grandement amélioré en stockant l’ensemble des agents formant la chaîne dans l’ordre de déplacement, cela permettrait d’empêcher les chaînes de former des cycles.