MIF16 - Techniques d'intelligence artificielle
Jeu du taquin

Hana Sebia - Tarik Boumaza

Juin 2021

1 Objectif

L'objectif de ce projet est l'implémentation d'une résolution d'un jeu de taquin (grille de 5×5) avec des agents intelligents.

2 Méthode

2.1 Grille

Nous avons créé une classe `Grille`, qui constitue la classe principale de notre application. Elle comporte principalement les différents agents ainsi que leurs positions respectives. Au lancement, une grille est donc générée en affectant aux agents des positions tirées aléatoirement.

2.2 Agents intelligents

Un agent, représenté par la classe `Agent` (qui hérite de la classe `Thread`), est composé de :

- sa position actuelle
- sa position finale
- sa messagerie (*cf.* section 2.3 Messagerie)

Les agents ont également connaissance de la grille, pour pouvoir déterminer quelles positions déjà sont occupées. Lorsqu'un agent veut se diriger vers sa position finale, il a 3 options testées dans l'ordre suivant :

1. Il peut y aller directement, de façon optimale (en ne passant que pas des cases libres).
2. Il peut y accéder en contournant des cases occupées.
3. Il ne peut y accéder par aucun chemin libre. Il demande alors à un agent se trouvant sur son chemin de se déplacer.

2.3 Messagerie

Pour qu'un agent puisse demander à un autre de libérer une case, nous avons implémenté la classe `Messagerie`, qui se compose d'un expéditeur, d'un destinataire et d'une case à libérer. Un agent destinataire d'un message tente de quitter la case à libérer (s'il se trouve sur la case à libérer) en se déplaçant sur une case libre autour de lui, ou en envoyant lui même un message à un voisin pour qu'il puisse se déplacer.

2.4 Déroulement

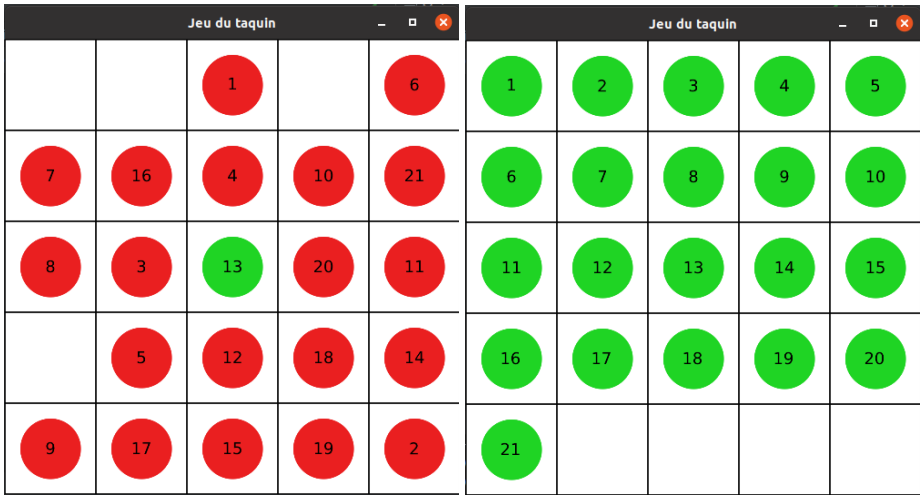
Au lancement de l'application, la grille est générée, on lance les threads (agents), et ces derniers coopérent alors pour résoudre le taquin. En effet, pour réduire les conflits (accès concurrents par ailleurs gérés avec la directive `synchronized`), nous les avons configurés de telle sorte que chacun d'entre eux ne tente d'atteindre sa position finale que lorsque la ligne au dessus de sa position finale est parfaitement constituée. En attendant, il se contente de coopérer et d'aider les autres agents à atteindre leurs objectifs.

2.5 JavaFX

Enfin, l'affichage graphique de la grille a été implémenté via *JavaFX*. Il est rafraîchi à chaque déplacement d'un agent intelligent.

3 Résultats

L'implémentation réalisée permet alors de résoudre les grilles 5 × 5 contenant 20 agents. Une grande majorité des grilles contenant 21 agents sont également résolues (plus de 90%).



(a) Entrée (b) Sortie

Figure 1: Grille 5 × 5 aléatoire avec 21 agents

4 Discussion

La résolution de la grille avec 20 agents est satisfaisante. On obtient un temps moyen de résolution de 30s.
Cependant, la résolution de la grille avec un nombre d'agents supérieur ou égal à 21 est plus difficile en suivant la stratégie implémentée (résolution ligne par ligne). En effet, la reconstitution de la dernière ligne doit être faite en même temps que l'avant dernière ligne. Une solution possible est de résoudre les deux dernières lignes colonne par colonne. Ainsi, la résolution se ferait en même temps.