Rapport Projet
TIA

Interaction Multi-Agents

Groupe:
NESR Mohamad
11811466

Enseignant: **AKNINE** Samir

Année universitaire **2021/2022**

Contexte

Dans le cadre de l'UE TIA, on a réalisé un Taquin dont son fonctionnement est basé sur les interactions multi-agents. Pour cela, nous avons fait en sorte que chaque case qu'on souhaite déplacer est un agent qui doit communiquer avec les autres cases afin de se déplacer. Chaque agent est donc un thread auquel une priorité est affectée afin de gérer ses mouvements.

Notre Code

Nous avons pris le choix de réaliser le projet en java avec l'éditeur IntelliJ et la librairie Java Swing pour l'interface graphique. Notre code est composé de 5 classes : Tableau, Agent, Position, Message et bien sûr la classe Main.

Agent

Nos agents sont des classes filles de la classe Thread. Ces agents sont initialisés avec une position initiale de départ, une position cible, un symbole et une image afin de les distinguer. Après l'exécution, ces agents se déplacent en fonction de leurs priorités afin d'atteindre leur position cible. Ce mouvement est fait avec un algorithme A* accompagné d'un comportement un peu aléatoire dans certains cas. Nous avons gerer le mouvement pour minimiser le temps de résolution de la façon suivante: si Xcible > Xactuel, l'agent se déplace vers le haut, si Xcible < Xactuel l'agent se déplace vers le bas, si Ycible > Yactuel, l'agent se déplace vers la droite, si Ycible < Yactuel l'agent se déplace vers la gauche. Également, quand notre agent atteint sa destination, sa priorité devient minimale pour que les agents qui n'ont toujours pas fini leurs mouvements deviennent plus prioritaires que lui.

Message

Afin de se déplacer, nos agents s'envoient des messages si la case qu'il veut atteindre est occupée par un autre agent. Ce message contient le symbole de l'agent, sa destination et le destinataire. Une fois le message reçu par l'autre agent, ceci cherchera une case libre autour de lui afin de libérer sa case actuelle à l'agent en mouvement. Si aucune case est libre, cet agent va à son tour envoyer un message à un de ses voisins afin de libérer une case et ainsi de suite.

Tableau

Cette classe contient notre plateau de jeu. On définit d'abord une taille de plateau (5 dans notre cas pour une meilleure visualisation) et le nombre d'agents qu'on veut placer dessus. Ensuite, on affiche notre plateau dans le terminal afin de visualiser nos résultats. Notre plateau s'imprime donc dans le terminal après chaque coup effectué. Nos cases libres sont représentées par le symbole "_" et nos cases occupées sont représentées par le symbole de l'agent à cette position.

Notre interface graphique est mise à jour simultanément avec notre affichage dans le terminal. On peut donc observer une simulation en temps réel des mouvements de nos agents vers leurs destinations finales.

Dans le rendu, l'affichage dans le terminal est commenté. Ceci est pour améliorer la performance et la vitesse de résolution du taquin.

Position

La classe position nous permet de récupérer les coordonnées d'une case et de faire une égalité entre deux positions différentes. Ceci nous facilite donc la comparaison de la position actuelle de notre agent à sa position cible.

Main

Dans la classe main, on initialise notre jeu et on permet à l'utilisateur de choisir son nombre d'agents souhaité avec la variable nbAgents. Ensuite, nbAgents seront sélectionnés de notre liste créée dans la classe Agents afin de les placer dans notre jeu.

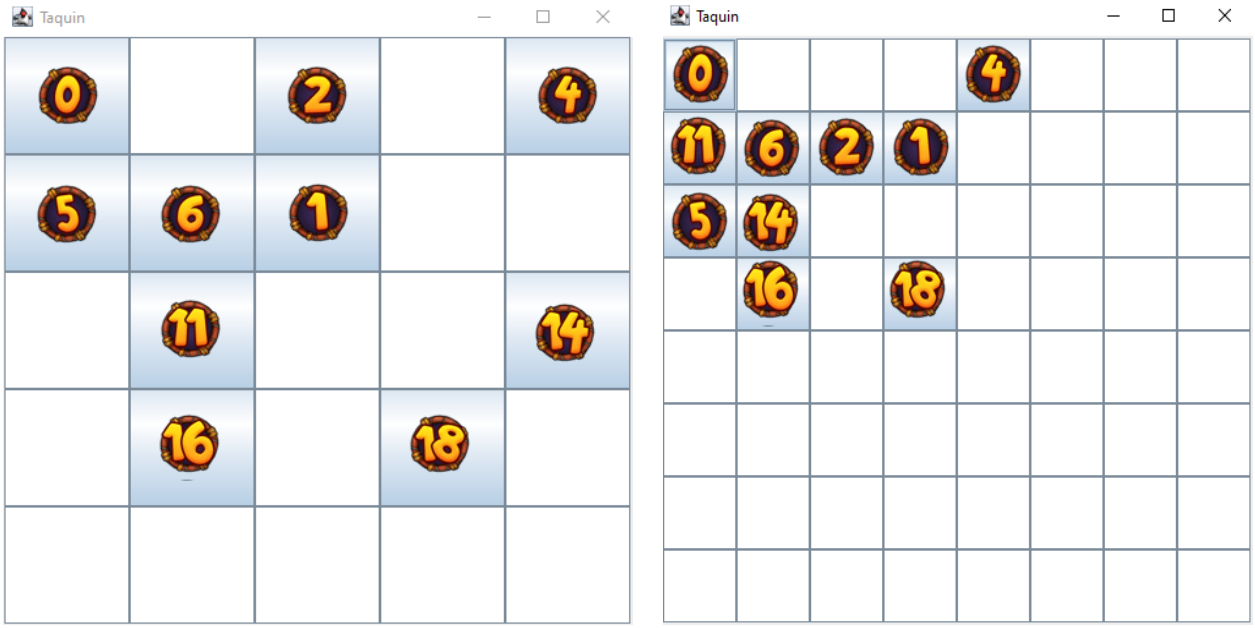
Resultats

Avec 12 agents, notre jeu se résout en environ 14 secondes. Avec 10 agents, le jeu se résout en 8 secondes. Avec 7 agents, le jeu se résout en 6 secondes. Avec 5 agents, le jeu se résout en 3 secondes. Bien sûr, ces valeurs sont des moyennes de plusieurs tests, dans certains cas avec 12 agents, le tableau se résout en 7 secondes et dans d'autres en 20 secondes.

On remarque clairement que notre résolution est rapide et efficace. Avec l'augmentation du nombre d'agents (plus de 12), la résolution devient compliquée due à

la puissance de mon ordinateur qui est un peu vieux et la gestion des threads que je trouve n'est pas faite de la manière la plus efficace (gestion naïve).

De même, notre grille est personnalisable, on a fait nos tests en 5*5 mais on aurait pu les faire en 8*8 si on voulais.



*Ceci n'est pas le résultat final de la résolution mais un exemple du jeu en pleine résolution de taquin en 5*5 et 8*8.*

Difficultés rencontrées

La plus grande difficulté dans ce tp était d'imaginer la structure du code et la gestion des threads. Une fois cela fait, le travail a été plutôt fluide.

Une deuxième contrainte lors du travail était la gestion du mouvement lors de la rencontre de deux agents. Dans certains cas, les mouvements effectués ne sont pas les plus optimales vu qu'une partie du mouvement est basée sur l'aléatoire.

De plus, la gestion des priorités des agents était de même un peu compliquée. Pour résoudre ce problème, nos agents qui sont des threads ont des priorités élevées qui seront minimisées quand ils atteignent leurs cibles afin de limiter leur mouvement et les rendre presque immobiles.

Pistes d'améliorations

Avec l'augmentation du nombre d'agents, la résolution met beaucoup plus de temps (plus de 12 agents). Ceci pourra être réglé en jouant d'une manière plus rigoureuse sur la priorité de nos agents afin de les déplacer d'une manière plus optimale. Une deuxième solution pourrait être l'implémentation d'un algorithme comme l'algorithme de Dijkstra afin d'en déduire le plus court chemin de chaque agent vers sa cible. On pourra donc vérifier si une résolution par algorithme de Dijkstra est meilleur qu'une résolution A* ou pas. Notre agent va donc suivre le trajet proposé par cet algorithme et ne sortira hors de son trajet que lors de conflits de positions avec les autres agents.

Une deuxième amélioration peut être la mise en place d'une personnalisation dans l'interface graphique au lieu du terminal. Dans notre cas, la sélection du nombre d'agents se fait dans le terminal. En ajoutant, une option de sélectionner ce nombre dans l'interface graphique, notre jeu pourra devenir plus interactif.

De plus, notre liste d'agents est codée en dur dans la classe Agent, une meilleure idée sera de coder une fonction qui génère aléatoirement les positions de départ et les positions cibles.

Conclusion

Ce TP nous a permis de nous familiariser avec les systèmes multi-agents. Ceci a été une première expérience pour nous avec les threads en java et de même, ça nous a permis de découvrir encore plus sur les systèmes multi-agents. C'est donc une bonne continuation des compétences développées lors du TP1 sur les systèmes multi agents avec une nouvelle approche différente et des objectifs différents. On considère ce TP comme une réussite malgré quelques lacunes et petits bugs dans notre code. Cependant, les portes d'améliorations restent ouvertes.

Demo

Dans la vidéo, nous avons l'impression qu'il y a des coupures dans l'affichage. Ceci est à cause de l'enregistrement d'écran sur ma machine qui est un peu ancienne. L'enregistrement d'écran n'arrive pas à se mettre à jour "frame par frame". Cependant, on peut toujours voir les mouvements un par un sur l'interface graphique en compilant le code.

demo.mp4