



# Technique d'IA

TP : Interaction Multi-Agents

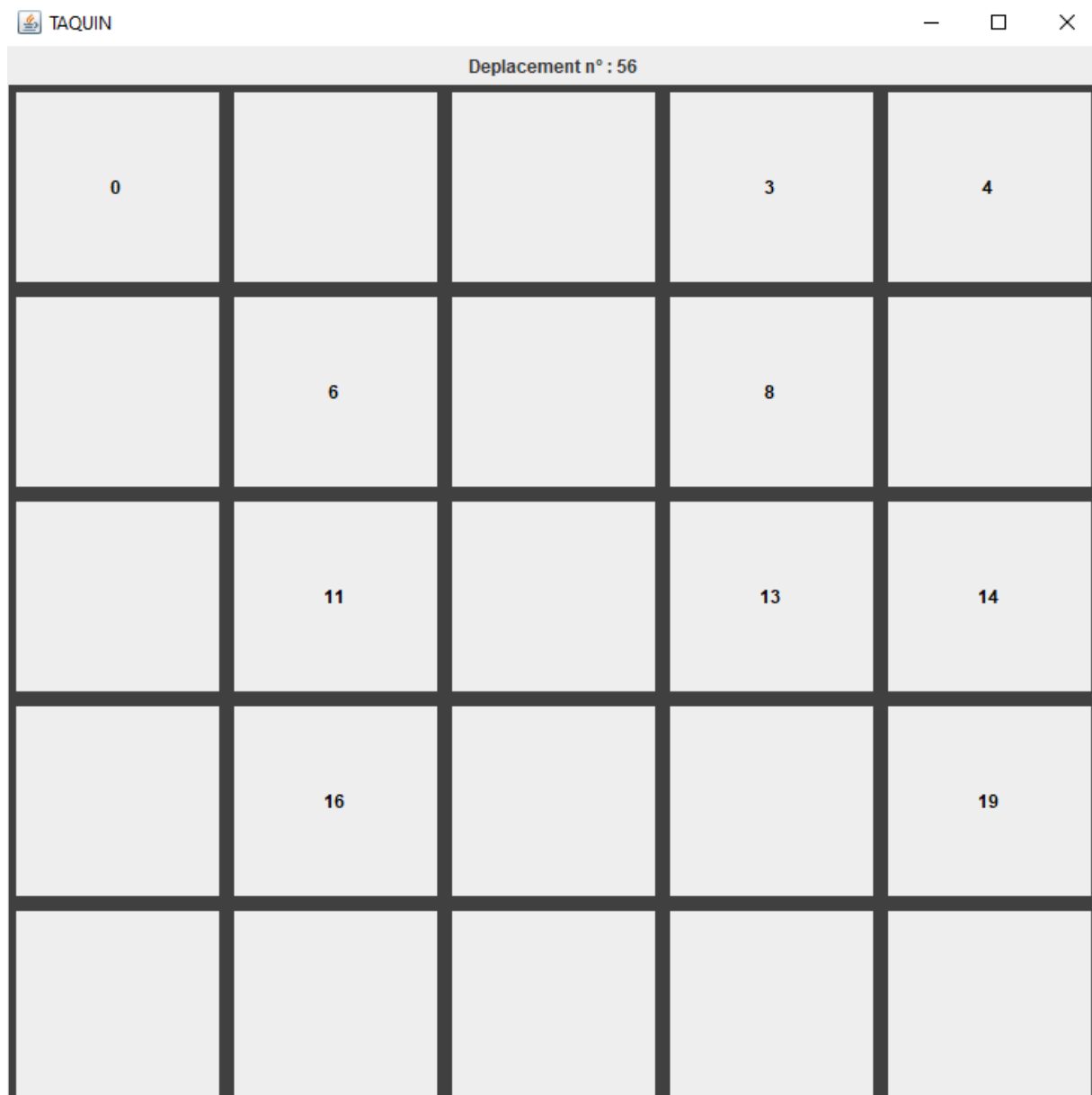
Yann VINCENT p1906701

Iliesse LARIBI p1911241

ENSEIGNANT : SAMIR AKNINE 2022-23

## Table des matières

Présentation générale .....	2
Notre façon d'implémenter.....	2
Notre code .....	2
Comportement des agents .....	3
Résultat .....	4
Problèmes restants .....	4
Conclusion.....	5



The image shows a screenshot of a software window titled "TAQUIN". The window contains a 5x5 grid of cells. The title bar of the window is "Déplacement n° : 56". The grid contains the following numbers:

Déplacement n° : 56				
0			3	4
	6		8	
	11		13	14
	16			19

## Présentation générale

Le TP consiste à réaliser un jeu du taquin qui se résout automatiquement. La taille du plateau est de 5\*5. Et le nombre de case a bien placer est variable.

Chaque case représente un agent et est donc gérée par un thread unique. Lorsqu'une case veut rejoindre sa position finale elle avance de case en case vers celle-ci. Si les cases menant à sa case finale sont vides pas de problème mais si un autre agent se trouve sur son chemin il faut les faire communiquer. L'agent qui veut passer envoie un message à l'agent sur son chemin en lui demandant de se décaler pour qu'elle puisse passer. Le jeu est fini lorsque tous les agents sont sur leur case finale.

## Notre façon d'implémenter

Au lancement du jeu on choisit le nombre d'agents (nombre de case a placé). Puis on regarde le jeu se résoudre tout seul. À la fin du jeu si le taquin a été résolu on a le nombre de mouvements qui a été effectué (il est aussi affiché dans la console et en haut de la fenêtre graphique en temps réel). La place finale d'un agent est définie à l'avance et est toujours la même pour chaque partie mais la place initiale d'un agent est aléatoire.

## Notre code

Le projet est divisé en 6 classes :

Agent.java : Classe qui gère le multithreading ainsi que le déroulement du jeu. Les différentes actions que doit effectuer un agent selon la situation ainsi que le cas d'arrêt du jeu.

Main.java : Initialisation du plateau de jeu des agents etc.

Position.java : class simple qui gère les positions x y des agents.

Message.java : class qui gère les messages qui seront envoyés entre les agents pour demander leurs déplacements.

Plateau.java : cette classe est la plus longue. Elle gère les cas de déplacement des agents. Par exemple les fonctions qui définiront quelle direction prendre pour rejoindre le plus rapidement la case finale.

Affichage.java : cette classe permet de gérer l'affichage graphique du jeu dans une fenêtre. Il existe aussi une version de l'affichage dans le terminal.

```

Agent[] tabAgents2 = {
    new Agent(tabpos[0], new Position(x:0,y:0), num:"0 "),
    //new Agent(tabpos[1], new Position(0,1), "1 "),
    //new Agent(tabpos[2], new Position(0,2), "2 "),
    new Agent(tabpos[3], new Position(x:0,y:3), num:"3 "),
    new Agent(tabpos[4], new Position(x:0,y:4), num:"4 "),
    //new Agent(tabpos[5], new Position(1,0), "5 "),
    new Agent(tabpos[6], new Position(x:1,y:1), num:"6 "),
    //new Agent(tabpos[7], new Position(1,2), "7 "),
    new Agent(tabpos[8], new Position(x:1,y:3), num:"8 "),
    //new Agent(tabpos[9], new Position(1,4), "9 "),
    //new Agent(tabpos[10], new Position(2,0), "10 "),
    new Agent(tabpos[11], new Position(x:2,y:1), num:"11 "),
    //new Agent(tabpos[12], new Position(2,2), "12 "),
    new Agent(tabpos[13], new Position(x:2,y:3), num:"13 "),
    new Agent(tabpos[14], new Position(x:2,y:4), num:"14 "),
    //new Agent(tabpos[15], new Position(3,0), "15 "),
    new Agent(tabpos[16], new Position(x:3,y:1), num:"16 "),
    //new Agent(tabpos[17], new Position(3,2), "17 "),
    //new Agent(tabpos[18], new Position(3,3), "18 "),
    new Agent(tabpos[19], new Position(x:3,y:4), num:"19 "),
    new Agent(tabpos[20], new Position(x:4,y:0), num:"20 "),
    new Agent(tabpos[21], new Position(x:4,y:1), num:"21 "),
    new Agent(tabpos[22], new Position(x:4,y:2), num:"22 "),
    new Agent(tabpos[23], new Position(x:4,y:3), num:"23 "),
    //new Agent(tabpos[24], new Position(4,4), "24 "),
};

```

Pour ajouter les agents que l'on souhaite il suffit de les décommenter dans le Main.java ligne 60. Au lancement du programme on choisit le nombre d'agents sur le Plateau et le programme prendra dans l'ordre les X premier agent décommenter.

Ici si on choisit 6 agents au début du programme, il devra placer les agents numéro : 0,3,4,6,8,11.

A noter que le programme fonctionne avec toutes les combinaisons d'agent possibles.

Certaines combinaisons seront simplement plus longues que

d'autre à être résolue (plus les cases d'arrivée des agents sont compactées au même endroit plus la résolution sera longue.

## Comportement des agents

Nous avons décidé de gérer les choix des agents sous forme de priorité.

Lorsqu'un agent veut se déplacer il regarde d'abord la distance qui le sépare de son arrivée en fonction de la prochaine case sur laquelle il ira. Il prend donc la case la plus proche de son arrivée et qui n'est pas la case sur laquelle il était à son dernier déplacement (case précédente) et regarde s'il peut y aller. C'est-à-dire si ce n'est pas en dehors du plateau et s'il n'y a pas déjà un agent dessus.

Cas suivant si la case la plus proche est occupée ou est la case précédente par exemple. On prend aléatoirement une case vide autour de lui qui est accessible et qui n'est pas la case précédente. Si ces conditions ne sont pas respectées alors l'agent commencera à regarder avec quel voisin autour de lui il voudra communiquer pour avancer. Il prendra la prochaine case la plus proche de son arrivée et demandera au voisin (en lui envoyant un message) de se déplacer pour le laisser passer.

Dernier cas, l'agent reçoit un message d'un autre agent lui indiquant qu'il doit se déplacer. Dans ce cas on reprend l'ordre des priorités expliqué ci-dessus mais avec une particularité, il est autorisé à se déplacer sur sa case précédente afin de ne pas bloquer le jeu.

## Résultat

Afin d'être le plus précis possible nous avons réalisé un tableau des résultats de notre programme. Nous avons réalisé 10 tirages aléatoire pour chaque nombre d'agents dans un tableau jusqu'à 12. Nous avons récupéré le taux de réussite et le nombre de coup moyen. Pour 13 agents 5 tirages, pour 14 et 15 agents, 2 tirages. Ensuite le nombre de déplacement était beaucoup trop grand et prenait trop de temps à se résoudre. Nous pouvons supposer que la suite fonctionne mais en beaucoup trop de mouvements.

NB Agent/Tentative n°	1	2	3	4	5	6	7	8	9	10	Moyenne	Pourcentage réussite
1	6	6	5	4	4	1	4	1	6	4	4,1	100%
2	11	6	12	5	6	9	5	3	4	6	6,7	100%
3	3	9	10	10	21	9	7	8	18	13	10,8	100%
4	19	17	7	14	17	10	16	18	25	14	15,7	100%
5	17	14	19	18	15	32	19	28	18	16	19,6	100%
6	27	20	31	17	137	32	45	24	76	18	42,7	100%
7	24	27	50	43	83	33	55	48	33	41	43,7	100%
8	66	51	72	56	41	87	45	97	65	108	68,8	100%
9	68	66	141	54	51	127	76	115	160	435	129,3	100%
10	117	184	109	326	169	220	98	825	150	80	227,8	100%
11	126	332	196	96	330	932	422	87	96	1271	388,8	100%
12	2912	709	368	111	4508	11646	1122	20000	2539	20158	6407,3	100%
13	1860	20000	14343	1297	50000						17500	
14	15087										15087	
15	9887										9887	

Les valeurs peuvent changer en fonctions des agents choisies. Comme dit précédemment choisir des agents avec des positions compactent dans le Plateau augmenterons le nombre de coups.

## Problèmes restants

Nous avons repéré 2 problèmes. Le premier étant l'optimisation. En théorie notre programme est capable de réussir tout type de plateau mais par moments le nombre de coups est beaucoup trop élevé (lorsque le nombre d'agents est élevé). Cela vient sûrement d'une décision aléatoire prise lors du choix de case libre (si la distance la plus courte entre sa position et sa position finale est occupée).

Le deuxième problème est un problème d'affichage. Par moment sur l'affichage (terminal et graphique) après la résolution il arrive rarement que certain agent se déplace encore d'une ou deux cases. Pour vérifier que le programme a belle et bien marché il suffit dans le terminal de remonter d'un ou deux affichages de plateau pour voir apparaître plusieurs fois la ligne « Nombre de déplacements finales : X » avec par-dessus le plateau final bel et bien compléter. Ce problème vient sûrement du cas d'arrêt de notre

programme qui se fait thread par thread. Mais malgré ce bug le plateau est bien résolu sans problème.

## Conclusion

Ce projet nous a beaucoup apportés sur la compréhension du multithreading et de la communication entre agents. Nous avons aussi trouvé intéressant de réfléchir à un ordre de priorité pour le comportement des agents.