Techniques de l'Intelligence Artificielle

Interaction Multi-Agents — Rapport de projet

Vichith LY

19 juin

1. Modélisation du problème

On cherche à modéliser un jeu de puzzle. Sur chaque case de ce puzzle, il peut y avoir un agent représenté par une valeur (symbole, lettre, etc.). Chaque agent possède une case de départ (case courante au début), et une case destination qu'il doit atteindre en se déplaçant de case en case. On considère un puzzle terminé lorsque chaque agent réussit à atteindre sa case destination.

1.1. Grille

On choisit de représenter l'environnement du jeu par une grille carrée de dimension N x N. On fixe N = 5 avec N le nombre de cases sur une ligne. Notre grille comporte ainsi 5 x 5 = 25 cases.

Classe ***Grid*** (grille)	
Attributs	• size (entier) : largeur / hauteur de la grille. • boxes (tableau 2D) : matrice contenant des objets de la classe ***Box.***

1.2. Case

Chaque case de notre grille doit posséder des coordonnées x et y pour qu'elle puisse être située dans l'environnement. Une case peut contenir un agent.

Classe ***Box*** (case)	
Attributs	• x (entier) : numéro de ligne dans la grille. • y (entier) : numéro de colonne dans la grille. • agent (***Agent***) : objet de la classe ***Agent.***

1.3. Agent

Un agent est une entité capable de se déplacer, seulement si la case sur laquelle il se dirige est vide (i.e. il n'y a pas d'autres agents dessus) et que cette case est à l'intérieur des limites de notre grille. On distingue 4 types de déplacement suivant les directions nord, sud, est et ouest.

Classe ***Agent***	
Attributs	• value (***Letter***) : valeur de l'agent, sous forme d'une lettre alphabétique. ○ ***Letter*** est une énumération de lettres, chaque lettre possède une valeur (e.g. A = 1, B = 2, etc.). • source (***Box***) : case de départ. • destination (***Box***) : case à atteindre. • current (***Box***) : case courante. • priority (entier) : indice de la lettre dans l'alphabet.
Méthodes	• isArrived() : retourne un booléen vrai si l'agent est arrivé à destination. • getNeighbours() : retourne une liste de ses agents voisins. • getFreeBoxNeighbours() : retourne une liste de cases voisines libres.

1.4. Message

Afin de reconstruire un puzzle dans n'importe quelle configuration d'agents, il est nécessaire de mettre en place une approche cognitive. Cette approche est basée sur l'échange de messages entre agents, afin qu'un agent puisse avancer au fur et à mesure vers sa destination. Par exemple, si un agent bloque le passage d'un autre agent, il lui enverra un message pour lui demander de libérer la case sur laquelle il veut se diriger.

Classe ***Mail***	

Attributs	• sender (***Agent***) : l'agent expéditeur.
	• receiver (***Agent***) : l'agent destinataire.
	• type : (***Type***) : le type de message.
	  ○ ***Type*** est une énumération. On distingues 2 types : REQUEST et RESPONSE.
	• content (***Content***) : le contenu du message.
	  ○ ***Content*** est une énumération de type. On distingues 3 types de contenu : MOVE (demande de déplacement), OK (demande effectuée), NOK (demande non effectuée).
	• priority (entier) : utile pour ajouter un ordre de traitement dans les messages ; sa valeur est égale à la priorité de l'expéditeur.

Sender	Receiver	Type	Content	Priority

Figure 1 — Modélisation d'un message (Mail)

1.5. Boite aux lettres

On choisit de modéliser une boîte aux lettres partagée par tous, permettant à chaque agent d'envoyer ou de recevoir un message de tous les autres agents.

Classe ***MailBox***	
Attributs	• mails (dictionnaire) : structure de donnée stockant les messages (***Mail***) reçus de chaque agent. On peut récupérer les mails (valeur) d'un agent en spécifiant sa clé dans le dictionnaire. • sent (dictionnaire) : structure de données stockant les messages (***Mail***) envoyés par chaque agent.
Méthodes	• récupérer les mails d'un agent • récupérer les mails envoyés d'un agent • récupérer / supprimer un mail d'un agent

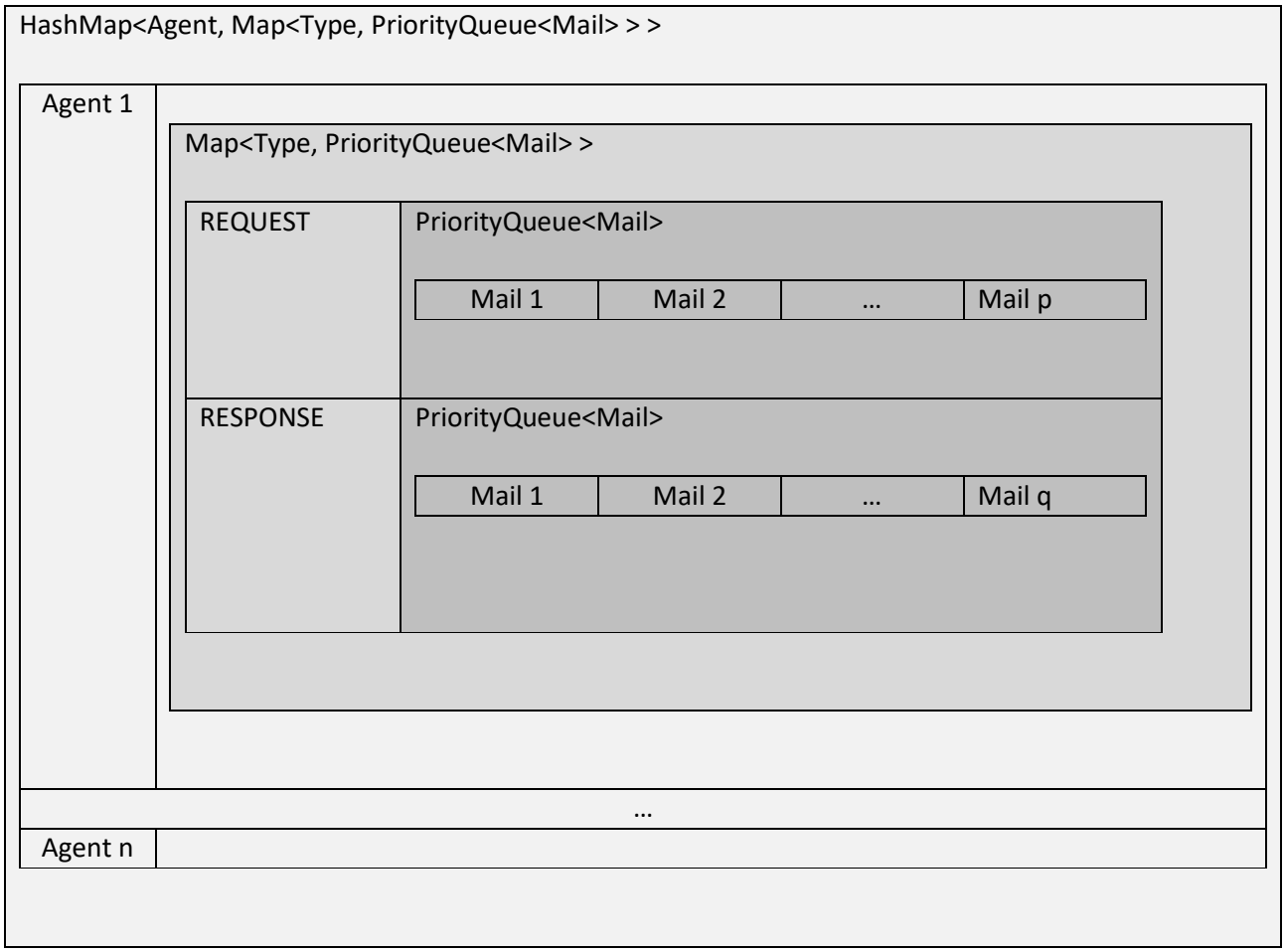


Figure 2 — Modélisation d'une boite aux lettres (MailBox) en Java

1.6. Multithreading

La résolution du puzzle se fait étape par étape. À chaque étape, un pool de n threads est exécuté. Un thread correspond à un agent et chaque agent possède une fonction de résolution appelée à chaque étape.

Tant que le puzzle n'est pas résolu, exécuter :	
Pool 1 de n threads	Agent_1.solve() … Agent_n.solve()
…	..
Pool p de n threads	Agent_1.solve() … Agent_n.solve()

Figure 3 – Exécution des pools de threads lors de la résolution du puzzle

1.7. Interface graphique

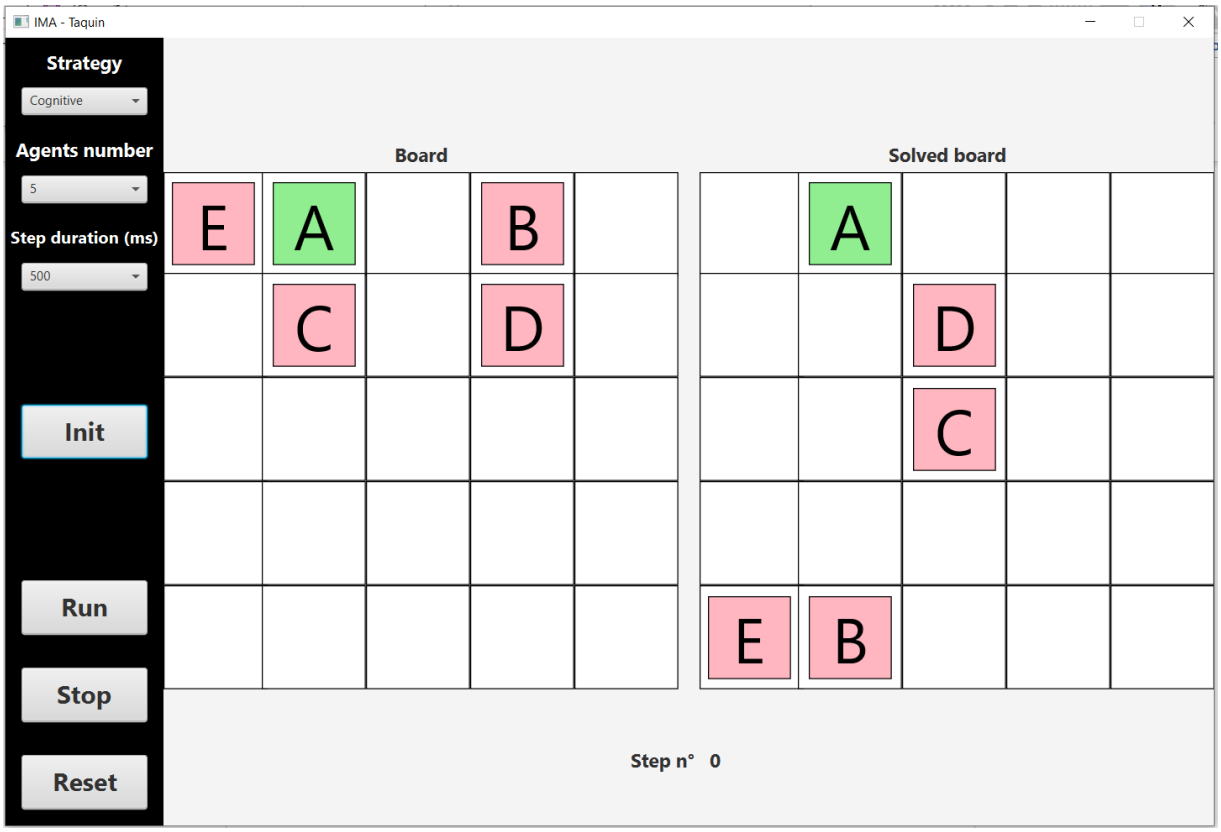


Figure 4 — Interface graphique du jeu de puzzle

1.7.1. Environnement

- Grille initiale (à gauche) : c'est dans cette grille que le puzzle se résout étape par étape.
- Grille finale (à droite) : configuration résolue de la grille du jeu.
- Agents : chaque agent est représenté par sa valeur (lettre) ; la couleur verte ou rouge indique si l'agent a atteint ou non sa destination.
- Compteur de nombre d'étapes (en bas des grilles) nécessaires à la résolution du puzzle.

1.7.2. Paramètres

- Choix de la stratégie de résolution : naïve, simple ou cognitive.
- Nombre d'agents dans le puzzle : de 1 à 24.
- Durée en ms entre chaque étape de résolution (1 étape = 1 exécution de n agents en multithread)

1.7.3. Actions

- Initialiser le jeu : positionner aléatoirement les agents à leur position initiale (grille de gauche) et à leur position finale (grille de droite).
- Lancer le jeu : tant que tous les agents ne sont pas arrivé à destination, continuer la résolution.

2. Stratégies de résolution

2.1. Approche naïve

Dans cette approche, les agents effectuent des déplacements uniquement sur la base de leur perception de l'occupation des cases voisines. À chaque étape, les agents font en sorte d'atteindre en premier la bonne ligne de sa destination, puis enfin la bonne colonne.

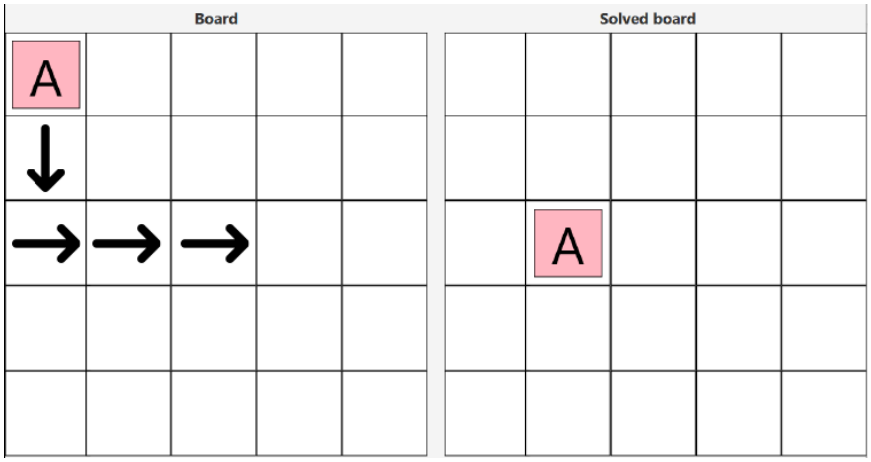


Figure 5 – Résolution naïve avec 1 agent : succès

Avec ce type de résolution, l'agent arrive à destination que s'il ne croise pas d'autres agents sur son chemin. Cette stratégie n'est donc pas du tout efficace lorsque le nombre d'agents croît.

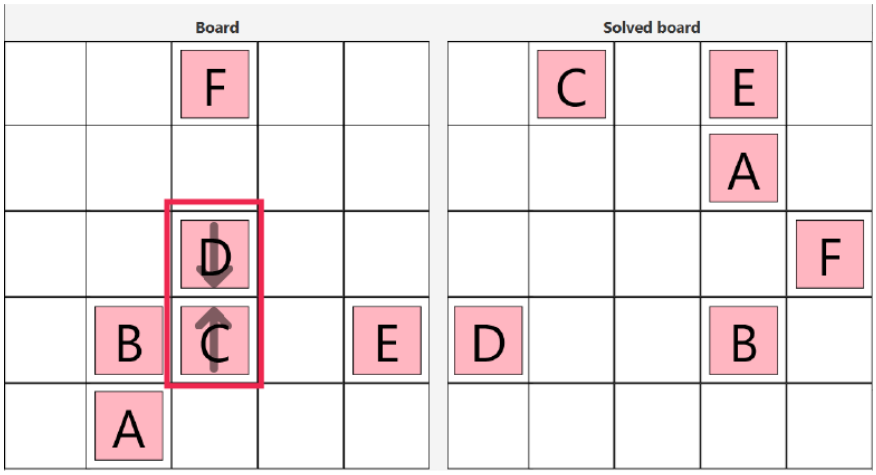


Figure 6 – Résolution naïve avec 6 agents : échec

2.2. Approche simple

Avec l'approche simple, les agents se déplacent uniquement après avoir identifié le meilleur chemin à emprunter vers leurs destinations. Pour cela, on utilise un algorithme de recherche de plus court chemin au sein d'une matrice, avec la méthode Breadth-first-search (BFS). Ainsi, l'algorithme permet à l'agent d'éviter les obstacles (les autres agents), et d'adapter sa trajectoire au fil des étapes de résolution.

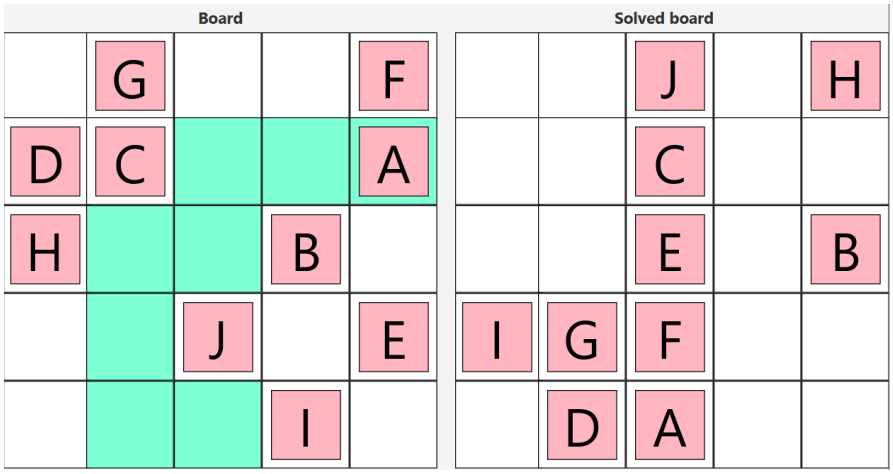


Figure 7 – Affichage du plus court chemin de l'agent A vers sa destination

Avec ce type de résolution, il est possible de résoudre des grilles avec au plus 11 agents suivant les configurations. Au-delà, certaines configurations peuvent créer des blocages.

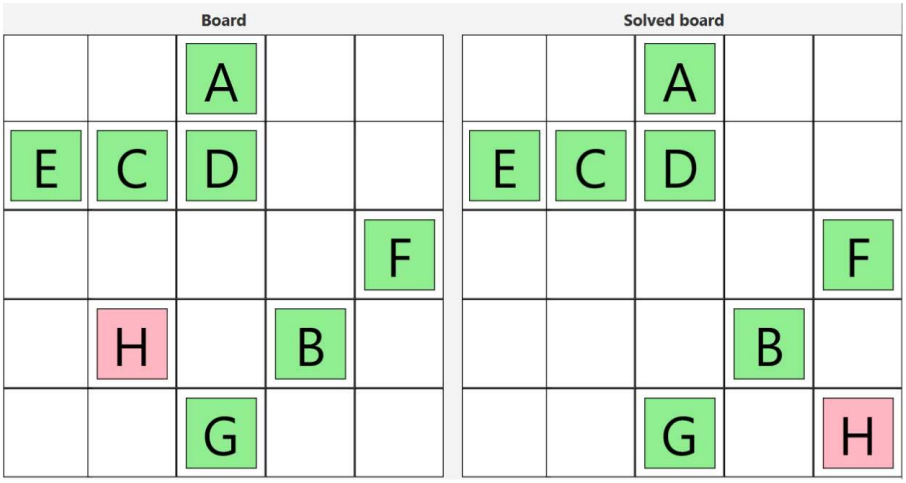


Figure 8 – Situation de blocage pour l'agent H

2.3. Approche cognitive

Cette approche utilise le calcul du plus court chemin de la stratégie simple, avec en plus un système d'interaction entre agent avec des messages (cf. ***MailBox*** et ***Mail***). Ainsi, lorsqu'un agent n'arrive plus à calculer un plus court chemin (i.e. est bloqué entre plusieurs agent), celui-ci lui envoie un message pour lui demander de se décaler.

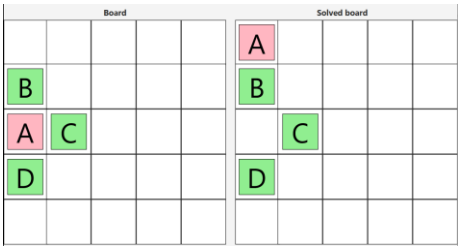
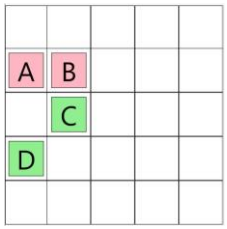
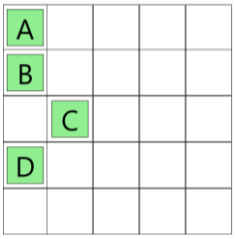
N° étape	État de la grille	Résolution
0		- A doit rejoindre sa destination dans le coin en haut à gauche de la grille, mais il est bloqué par 3 autres agents déjà arrivés. - A envoie un message à B, car il est sur son chemin.
1		- B reçoit le message de A qui lui demande de se déplacer. - B se déplace sur une case qui n'est pas sur le chemin de A et envoie une réponse positive à A. - A peut se déplacer sur la case qui vient de se libérer.
3		- A rejoint sa case destination. - B retourne vers sa case destination. Le puzzle est résolu.

Figure 9 – Exemple de résolution avec une approche cognitive

Avec ce type de résolution, il est possible de résoudre des grilles avec au plus 20 agents avec cependant un nombre d'étapes relativement élevé (> 1000).

3. Conclusion

Une approche cognitive à la résolution d'un puzzle a pu être implémentée grâce à un système d'envoi de messages et la définition de structures adéquates. Néanmoins, son implémentation mériterait d'être optimisée ou revue pour atteindre une résolution avec 24 agents. En l'état actuel, les calculs de l'approche cognitive sont très gourmands en ressources mémoire lorsque le nombre d'agents tend vers 20.