Rapport du projet Dedale

UE FoSyMa

Zeyu TAO Jiahua LI
Groupe 7

Master AI2D
Sorbonne Université

Mai 2025

Rapport FoSyMA

Contents

1 Introduction

L'objectif du projet est d'implémenter un système multi-agents sur la plateforme JADE, conçu pour une variante du jeu **Hunt the Wumpus**. Nous disposons d'un ensemble d'agents coopératifs dont le but est d'explorer un environnement inconnu et de collecter la plus grande quantité possible de trésors. L'environnement comporte également des agents adverses: **Golems**, qui complexifient la tâche des agents.

2 Architecture et implémentation

2.1 FSMBehaviour

Nous avons utilisé **FSMBehaviour** pour implémenter les comportements des trois types d'agents. FSMBehaviour représente un automate à états finis, dont chaque état correspond à une instance qui hérite de la classe **Behaviour**. FSMBehaviour permet une exécution séquentielle des behaviours, cette approche permet ainsi un traitement plus structuré et prévisible, tout en clarifiant la séquence d'actions exécutées et en simplifiant la gestion d'interactions entre les différents behaviours.

Nous avons commencé par définir un automate de base, illustré en Fig.1, qui constitue le composant de base pour la construction d'automates plus grands et plus complexes.

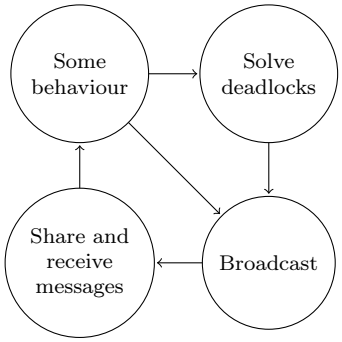

Figure 1: Composant de base pour la construction d'automates plus grands et plus complexes. Le nœud "Some behaviour" peut représenter n'importe quel behaviour impliquant un déplacement ou TankerAgentBehaviour.

Nous avons ensuite construit un nouvel automate décrivant les comportements complets des agents collecteurs, en combinant plusieurs instances du composant de base présenté ci-dessus (Fig.1). Notons que l'automate associé aux agents explorateurs est très similaire à celui des collecteurs, à la différence près qu'il ne comporte pas les états **Collect trésors** et **Empty pack**.

L'automate correspondant aux agents silo est identique à celui de la Fig.1, à l'exception de l'état **Solve deadlocks**, qui a été supprimé.

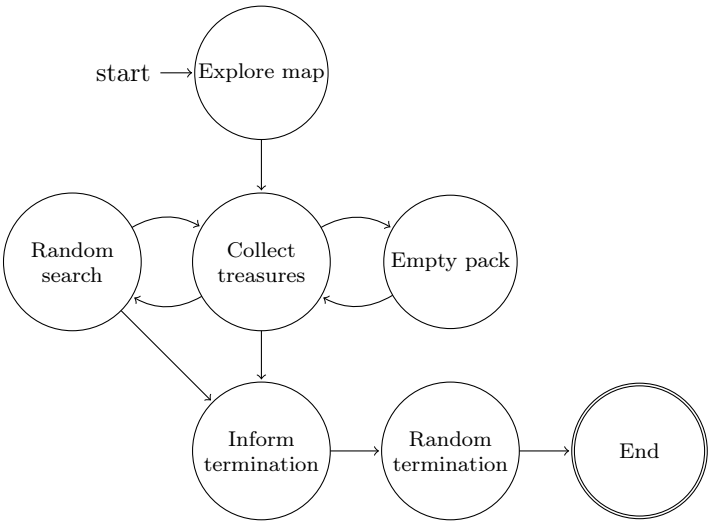

Figure 2: Automate associé aux agents collecteurs. Chaque état correspond à une instance de l'automate de la Fig.1, dans laquelle l'état "Some behaviour" a été remplacé par un behaviour concret.

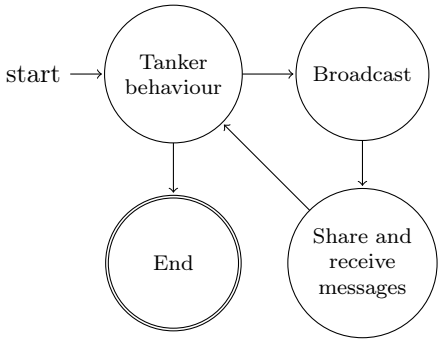

Figure 3: Automate associé aux agents silo.

2.2 Stratégies développées

2.2.1 Protocole de communication

La communication entre les agents suit le protocole illustré à la Fig.4. Les différentes étapes sont détaillées ci-dessous:

- **BROADCAST**: à chaque étape, l'agent envoie un message **BROADCAST** uniquement aux agents pour lesquels il dispose de nouvelles connaissances à partager. S'il n'a aucune connaissance à transmettre, il ne communique pas. Cette approche permet de limiter les partages inutiles, dans lesquels l'agent recevrait des connaissances qu'il possède déjà.
- **BROADCAST-ACK** : lorsqu'un agent reçoit un message **BROADCAST**, il renvoie un accusé de réception **BROADCAST-ACK** à l'agent qui a envoyé le message.

- **DATA-SHARE**: lorsqu'un agent reçoit un accusé de réception **BROADCAST-ACK**, il renvoie en retour un message **DATA-SHARE**, accompagné d'un contenu constitué d'un objet sérialisable: **DataShare**. Cet objet regroupe l'ensemble des connaissances nécessaires au jeu, telles que les informations sur la carte, les trésors, les agents silo, le nombre de trésors collectés par chaque agent, ainsi que les préférences des agents.
- **DATA-SHARE-ACK**: lorsqu'un agent reçoit un message **DATA-SHARE**, il met à jour ses connaissances, puis envoie un accusé de réception **DATA-SHARE-ACK**. De plus, lorsqu'un agent reçoit un accusé de réception **DATA-SHARE-ACK**, il met à jour ses connaissances concernant les partages effectués.

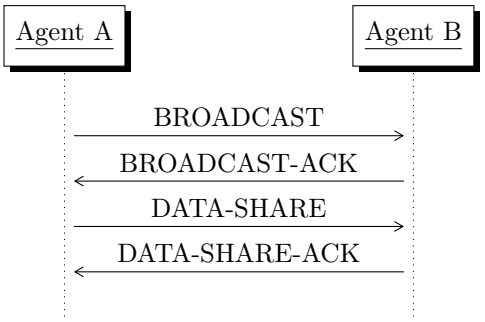

Figure 4: Diagramme du protocole de communication entre deux agents.

2.2.2 Déplacement des agents

Le déplacement des agents explorateurs et collecteurs repose sur le calcul du plus court chemin entre leur position courante et leur destination. Notons que les agents silo restent immobiles tout au long du jeu. Cette stratégie simple permet aux deux autres types d'agents de connaître toutes les positions des agents silo à la fin de la phase d'exploration.

À chaque étape, chaque agent dispose d'une liste de candidats, qui est une liste de positions vers lesquelles il souhaite se déplacer, ainsi que d'une liste d'arêtes interdites à utiliser pour le calcul du plus court chemin. Cette dernière contient en particulier, toutes les arêtes adjacentes aux agents silo, puisqu'il ne peut pas se déplacer vers une position occupée par un agent. Pour déterminer la prochaine position à emprunter, si l'agent dispose déjà d'un plus court chemin calculé, il lui suffit de le suivre jusqu'à atteindre sa destination. Sinon, il calcule un nouveau plus court chemin vers l'une des positions présentes dans sa liste de candidats (notons que cette liste est recalculée à chaque nouveau calcul du plus court chemin).

L'agent commence par calcule le plus court chemin à l'aide de l'algorithme de Dijkstra fourni par **graphstream**, puis il sélectionne la position ayant la plus petite distance calculée par Dijkstra. Enfin, il construit le plus court chemin entre sa position courante et la position sélectionnée.

L'implémentation de l'algorithme de Dijkstra fourni par **graphstream** a une complexité de $O(n \log n + m)$, où n est le nombre de nœuds et m le nombre

d'arêtes. La sélection de la position ayant la plus petite distance parmi la liste de candidats S a une complexité de $O(|S|)$, et la reconstruction du plus court chemin peut prendre au plus n nœuds, ce qui donne une complexité de $O(n)$. La complexité totale de cette étape est donc de $O(n \log n + m)$.

Nous avons utilisé cette approche pour tous les behaviours impliquant un déplacement, puisqu'elle est à la fois efficace et facile à étendre: il suffit de modifier la méthode de calcul de la liste de candidats pour l'adapter aux différents behaviours. Nous présentons ci-dessous la construction de la liste de candidats selon les différents cas de bahaviour:

- Explore Map: la liste de candidats est constituée des positions de tous les nœuds ouverts.
- Collect Treasure: la liste de candidats est constituée des positions des trésors disponibles dans l'environnement.
- Empty Pack: la liste de candidats est constituée de toutes les positions adjacentes aux agents silo.
- Random Search: la liste de candidats est constituée de k positions tirées aléatoirement, ainsi qu'une position adjacente à chacun des agents silo. Cette construction permet aux agents d'explorer l'environnement tout en revenant régulièrement vers les agents silo afin de échanger leurs connaissances.
- Termination Inform: la liste de candidats est constituée de toutes les positions adjacentes aux agents silo auxquels l'agent n'a pas encore informé sa terminaision.
- Random Termination: la liste de candidats est constituée de k positions tirées aléatoirement, avec k qui est au moins aussi grand que le nombre total d'agents présents dans l'environnement. Cette construction permet de garantir qu'il n'y a pas deux ou plusieurs agents qui tentent de se terminer à la même position.

2.2.3 Gestion des interblocages

La gestion des interblocages se déroule en trois étapes principales. Lorsqu'un agent se trouve dans une situation d'interblocage, il commence par effectuer une attente de t millisecondes, où t est une valeur tirée aléatoirement dans l'intervalle $[50, t_{max}]$ (dans le projet nous fixons $t_{max} = 1000$ ms). Ce délai aléatoire permet de décaler légèrement le moment d'action des agents afin de réduire les risques d'interblocage. Une fois ce délai écoulé, l'agent tente à nouveau de se déplacer vers la position où son dernier déplacement a échoué. Si cette position n'est occupée par aucun autre agent, alors il se déplace et termine la phase d'interblocage. Dans le cas contraire, il passe à l'étape suivante.

La deuxième étape consiste à recalculer un plus court chemin vers une nouvelle destination, c'est-à-dire, une position présente dans la liste de candidats (qui peut éventuellement être la même que précédemment). Si un tel chemin existe, alors l'agent se déplace et termine la phase d'interblocage. Sinon, il passe à la dernière étape. Notons que cette étape est la plus coûteuse en termes de calcul, avec une complexité de $O(n \log n + m)$.

Enfin, si les deux étapes précédentes ne permettent toujours pas de résoudre l'interblocage, l'agent effectue un déplacement aléatoire vers une position adjacente non occupée par un autre agent, puis passe à la phase de communication. Si l'interblocage persiste, l'agent relance une nouvelle tentative en répétant l'ensemble du processus, jusqu'à ce que l'interblocage soit résolu.

Cette approche présente certains inconvénients, notamment le fait que, dans certaines configurations ou situations, le nombre de tentatives nécessaires pour résoudre l'interblocage peut être arbitrairement grand, chaque tentative ayant une complexité de $O(n \log n + m)$, où n est le nombre de nœuds et m le nombre d'arêtes.

2.2.4 Ramassage des trésors

Le ramassage des trésors repose sur le principe suivant: dans un premier temps, l'agent se dirige vers le trésor de sa préférence le plus proche, en calculant un plus court chemin pour l'atteindre. Une fois arrivé à une position où se trouve un tel trésor, il le ramasse, puis va directement vers un agent silo pour vider son sac. Cette approche constitue une première implémentation simple, mais peu efficace. Malheureusement, faute de temps, nous n'avons pas pu mettre en place une version optimisée. Une amélioration envisageable serait de permettre à l'agent de continuer à ramasser des trésors de sa préférence tant que sa capacité le permet, avant d'aller vers un agent silo. Notons qu'il s'agirait alors d'un algorithme glouton classique, où l'agent cherche à maximiser la quantité de trésors ramassés tout en minimisant les distances parcourues.

Chaque agent possède une connaissance du nombre total de trésors initialement présents dans l'environnement (notons que cette valeur ne correspond pas du tout à la quantité de trésors présents dans l'environnement). Ainsi que du nombre total de trésors ramassés par chaque agent. À la fin de la phase de collection, si le nombre total de trésors ramassés n'atteint pas celui des trésors initialement présents, l'agent effectue alors une recherche aléatoire pour réexplorer la carte afin de trouver les trésors manquants et de les ramasser.

Cette approche exige que tous les agents connaissent exactement le nombre de trésors initialement présents dans l'environnement à la fin de la phase d'exploration. Cependant, elle n'est plus adaptée lorsque la position des trésors peut changer au cours de l'exploration, notamment si les trésors ne sont pas enfermés dans un coffre ou en présence d'agents adverses particuliers capables d'ouvrir les coffres.

2.2.5 Ramassage collectif des trésors

Dans le cadre du projet, les trésors peuvent être ramassés de manière collective, notamment lorsqu'un agent ne dispose pas de l'expertise nécessaire pour ouvrir et récupérer le trésor concerné. Nous avons envisagé un protocole pour répondre à ce problème, mais, en raison de contraintes de temps, nous n'avons pas pu le mettre en œuvre.

Ce protocole se déroule de la manière suivante: lorsqu'un agent (notons A) arrive à une position de trésor et constate que son expertise ne lui permet pas de ramasser le trésor. L'agent A effectue alors une attente de t millisecondes,

cette durée t varie en fonction de la quantité de trésor présente à cette position, plus cette quantité est importante, plus t est élevé. En parallèle, il envoie un message **BROADCAST-HELP** à tous les agents, accompagné d'une liste de positions adjacentes.

Soit B un autre agent présent dans l'environnement et reçoit le message envoyé par l'agent A. Après réception du message, l'agent B calcule un plus court chemin vers l'une des positions indiquées dans liste fournie. Une fois que l'agent B atteint la position, il envoie un message **BROADCAST-HELP-ACK** à l'agent A, accompagné de son expertise, et enregistre l'agent A comme étant son parent (notons que nous cherchons à construire un arbre dont la racine est l'agent situé à la position de trésor, et la liste de positions est constituée des positions adjacentes des feuilles de l'arbre, afin qu'un nouvel agent puisse rejoindre cet arbre).

Ensuite, l'agent A reçoit le message **BROADCAST-HELP-ACK** de l'agent B, et de manière identique, il enregistre B comme étant son enfant. L'agent A recalcule la valeur de son expertise en ajoutant l'expertise de l'agent B. Si la nouvelle expertise lui permet de ramasser le trésor, il le ramasse et envoie un message **END-HELP** à tous ses enfants pour indiquer la fin du protocole (son enfant transmet ensuite ce message à ses propres enfants, et ainsi de suite, jusqu'à ce que tous les agents dans l'arbre aient reçu ce message). Sinon, les agents A et B effectuent une nouvelle attente de t millisecondes et poursuivent le processus de la même manière que précédemment.

Ce protocole se termine lorsque le trésor a été ramassé ou que le délai de t millisecondes s'est écoulé.

2.2.6 Terminaison des agents

Lorsque tous les trésors ont été ramassés, les agents explorateurs et collecteurs se dirigent vers les agents silo pour informer de la terminaison en envoyant un message **TERMINATION**. Une fois qu'un agent a réussi à informer tous les agents silo de sa terminaison, il effectue un dernier déplacement aléatoire vers une position avant de finir. Chaque déplacement vers un agent silo a une complexité de $O(n \log n + m)$ (le coût du calcul du plus court chemin). Et soit s le nombre d'agents silo présents dans l'environnement, la complexité totale (sans tenir compte de la situation d'interblocage) est donc $O(s(n \log n + m))$.

Les agents silo, quant à eux, reçoivent le message de terminaison, marquent l'agent ayant envoyé le message comme terminé, puis lui renvoient un accusé de réception **TERMINATION-ACK**. Une fois que tous les agents non-silo sont marqués comme terminés, l'agent silo peut alors se terminer à son tour.

3 Conclusion

3.1 Synthèse

Nous avons réalisé la plupart des tâches demandées, à l'exception du comportement des agents collecteurs pour le ramassage collectif des trésors. Notre système est capable d'explorer l'environnement, de ramasser tous les trésors

présents (sans garantie d'optimalité), de gérer la présence de plusieurs agents silo et de terminer correctement.

3.2 Regard critique et des améliorations possibles

Nous remarquons que notre système est très sensible à la position initiale des agents silo. Si les agents silo sont mal placés, par exemple lorsqu'un (ou plusieurs) agent silo se déplace vers une position qui découpe le graphe en plusieurs composants connexes, il peut y avoir des composants qui ne contiennent aucun agent. Dans ce cas, cette partie du graphe ne peut jamais être explorée, puisque nous avons décidé que les agents silo restent immobiles tout au long du jeu, et donc les agents explorateurs et collecteurs ne terminent jamais la phase d'exploration. Un autre problème survient lorsqu'un agent silo se trouve sur une position où se trouve un trésor. Dans ce cas, le trésor ne pourra jamais être ramassé par les agents, empêchant ainsi la phase de collecte de se terminer, puisque notre condition d'arrêt stipule que les agents doivent ramasser tous les trésors présents dans l'environnement. Actuellement, notre système n'est pas capable de détecter et gérer ces situations et nécessite une intervention manuelle pour arrêter. Une amélioration envisageable pour ce problème serait de trouver un moyen plus efficace de placer les agents silo, en utilisant par exemple un algorithme approprié. Une autre option serait de permettre aux agents silo de se déplacer lorsqu'un agent envoie une demande par message, donc il serait également nécessaire de mettre en place un mécanisme pour détecter ces situations.

Un autre problème est l'usage du temps machine comme timestamp du trésor. Cette méthode n'est pas appropriée dans un système distribué, car elle ne garantit pas une cohérence temporelle globale. Une amélioration envisageable serait l'intégration d'un système d'horloges vectorielles ou matricielles, mieux adaptées à la gestion des événements dans un environnement distribué.