Dedale-FoSyMa

Groupe 4 : Decker Benjamin & Guetteville Nathan

May 2025

1 Introduction

1.1 Rappel du sujet

Le but du projet est de programmer une série d'agents coopératifs capables d'évoluer dans un environnement dont ils ne connaissent pas la topologie initialement. Leur but est d'explorer cet environnement et de collecter un maximum de ressources (*or* ou *diamants*) réparties sur la carte. Plusieurs contraintes viennent limiter les actions des agents.
Premièrement, les agents peuvent être des *silos* ou des *collecteurs*. Les *silos* disposent d'une capacité de stockage illimitée, mais ne peuvent pas ramasser les ressources par eux mêmes. Les *collecteurs* possèdent un sac à dos de capacité limitée. Chacun de ces agents peut explorer l'environnement afin d'augmenter ses connaissances et collecter des trésors. Ces agents possédent des compétences différentes : la serrurerie (lockpicking) nécessaire pour ouvrir les coffres avant la collecte d'un trésor et la force (strenght) nécessaire pour le ramassage d'un trésor.
De plus, les agents sont capables de communiquer entre eux mais leur rayon de communication est limité et le nombre de messages envoyés devra être controlé.
Enfin, un agent adverse (le Golem) se déplace dans l'environnement et est capable de déplacer les trésors (entrainant alors une perte de ressources), de refermer des coffres ouverts et de bloquer les autres agents.

1.2 Généralités d'implémentation

Ce projet a été implémenté en utilisant la plateforme multi-agents JADE et plus particulièrement l'environnement Dedale. De plus, nous avons choisi de programmer les comportements de nos agents en utilisant un FSM afin de faciliter l'ajout au fur et à mesure de nouveaux comportements et transitions entre ces comportements ainsi que pour faciliter la compréhension de ces comportements via des représentations schématiques plutôt que par la simple lecture de notre code.

2 Exploration

L'exploration est implémentée dans le fichier $ExplorationBehaviour.java$, comme un $OneShotBehaviour$. Seuls les agents $collecteurs$ y participent. Ils parcourent le graphe selon un parcours en profondeur en se déplaçant vers les noeuds ouverts restant les plus proches afin de minimiser le nombre total de déplacement. À chaque déplacement, les agents mettent à jour leur carte en ajoutant les nouveaux noeuds découverts comme des noeuds ouverts et en fermant le noeud qui vient d'être visité. De plus, si un trésor est découvert, sa position et ses informations sont sauvegardés dans une liste $recordedTreasures$ qui est un attribut du $FSMCoopBehaviour.java$. L'exploration se termine lorsque tous les noeuds ont été visités. À la fin de l'execution de la méthode $action$, si aucun trésor n'a été trouvé, le FSM effectue la transition vers le comportement $MessageBehaviour$ qui permet le partage de cartes.

3 Communications

Les communications sont implémentées dans les fichiers $MessageBehaviour.java$, $PingBehaviour.java$, $PongBehaviour.java$ et $ShareMapsBehaviour.java$. Dans un premier temps, on cherche à déterminer si l'agent est à portée de communication d'un autre agent. Le $MessageBehaviour$ vérifie si l'agent a déjà envoyé un ping. Si c'est le cas, il attend un éventuel pong de réponse indiquant qu'un autre agent est à portée et souhaite échanger des informations. Dans ce cas, le FSM effectue la transition vers le $ShareMapsBehaviour$. Sinon si l'agent n'a pas envoyé de ping, le FSM bascule vers le $PingBehaviour$ qui envoie un message de protocole $PING$ à tous les autres agents qui le recevront s'ils sont à portée et met à jour le booleen $pingSent$ du FSM à $true$, puis repasse dans le $MessageBehaviour$. Enfin, si l'agent a reçu un ping alors il renvoie un message de protocole $PONG$ à l'émeteur du ping grâce au $PongBehaviour$ et passe ensuite au $ShareMapsBehaviour$. Ce protocole de handshake envoie $n = nombred'agents + 1$ messages.
Maintenant que les agents sont sûrs d'avoir un interlocuteur, dans le $ShareMapsBehaviour$, les agents cherchent tout d'abord à mettre à jour leur connaissance de la position du silo avec le protocole $SHARE-POS$, en utilisant un système d'horloge logique de Lamport. Cela permet aux agents d'avoir accès à la dernière position ou destination connue du silo.
C'est également dans ce dernier comportement que les cartes sont échangées. Les cartes sont partagées grâce au protocole $SHARE-TOPO$. Les agents n'échangent pas l'intégralité de leur carte. En effet, chaque agent a sa carte et une autre carte par agent au total. Lors de l'exploration, les nouveaux noeuds découverts sont ajoutés à toutes les cartes et lors de l'échange, les correspondants s'échangent leur carte respectives associée à leur interlocuteur puis les réinitialisent à zéro. Cela assure que seuls les noeuds qui n'ont encore jamais été partagé avec le correspondant en question le sont pour éviter toute redondance. Quatre messages sont envoyés lors de cet échange d'informations.

Enfin, pour ce qui est de la gestion des interblocages, ils sont tout d'abord détectés dans les comportement $ExplorationBehaviour$, $MoveToSiloBehaviour$ et $MoveSiloBehaviour$ après deux tentatives de mouvement infructueuses. Ils sont ensuite gérés par les comportements $UnblockBehaviour$ et $UnblockSiloBehaviour$ dans lesquels les deux agents qui se bloquent s'échangent leur position et leur priorité. La priorité des agents est basée sur leur capacité de sac à dos totale sauf pour le silo qui a une priorité négative. L'agent avec la plus faible priorité fait un mouvement aléatoire autre que celui qu'il voulait faire à la base tandis que l'agent avec la plus forte priorité reprend son exécution là où il en était avant d'être bloqué. Il y a donc deux messages envoyés par situation d'interblocage.

4 Collecte

Pour la collecte il y a d'abord un système passif : au cours de son exploration, un agent pourra être amené à se trouver sur un noeud contenant un trésor. Si c'est le cas, il rentre dans le Behaviour $CollectBehviour$ dans celui-ci il va récupérer toutes les informations du butin en essayant d'ouvrir le coffre et d'en prendre le contenu. Il stocke les informations du trésor à la fois dans la liste $recordedTreasures$ de son FSM et dans une structure spéciale : $HelpNeededForTreasure$ qui sera utilisé pour aller chercher de l'aide. S'il arrive à récupérer le trésor en entier, il le supprime de la liste $RecordedTreasures$; s'il en récupère une partie, il se dirige vers le silo (via $MoveToSilo$ et toujours avec un passage par $MessageBehaviour$ après chaque déplacement) pour y déposer ses trouvailles s'il en connaît la position, sinon il repasse en mode exploration en attendant d'obtenir cette information; s'il échoue à récupérer la moindre ressource, il repasse en mode exploration après avoir sauvegardé son $HelpNeededForTreasure$ dans son FSM, en espérant croiser un autre agent aux capacités adéquates. Lorsqu'un agent porteur d'un $HelpNeededForTreasure$ en croise un autre, après l'échange de cartes, il lui envoie son objet et le destinataire va s'occuper de prendre les décisions : s'il a les capacités manquantes, il va suivre le envoyeur. De plus si le destinataire est du type correspondant au trésor et que l'envoyeur ne l'est pas ou que la capacité de stockage du destinataire est plus élevée que celle de l'envoyeur, le destinataire endosse le rôle de leader, c'est-à-dire que c'est lui qui va se rendre (via $MoveToTreasure$et toujours avec un passage par $MessageBehaviour$ après chaque déplacement) sur le noeud en question pour ouvrir et récupérer le trésor, tandis que l'autre s'arrêtera à proximité. Sinon, les rôles s'inversent. Si le destinataire ne peux pas aider du tout, les deux agents retournent à leur exploration.

Quand un agent a fermé tous ses noeuds, il passe en collecte dite "active" en se servant de la liste $RecordedTreasures$ pour se diriger vers le trésor le plus proche et enchaîner ainsi les trésors. Dans cette phase, lorsque la liste de trésors et vide, l'agent arrête son exécution.