Rapport de projet - Systèmes Multi-Agents

MinerAgent

M1 Informatique - S2 2021

Enseignante : Nadia Kabachi

Titouan Knockaert & Gaspard Goupy

Sommaire

Présentation du projet

Ce projet a été réalisé dans le cadre de l'enseignement *Systèmes Multi-Agents* du M2 IA à l'université Claude Bernard Lyon 1. Plus précisément, il a été encadré par Mme. Kabachi et s'intéresse à la modélisation *Belief-Desire-Intention* (*BDI*). L'objectif était de réaliser une simulation multi-agents *BDI* en Java, avec le Framework *JADE*.
La simulation reproduit le scénario suivant : un vaisseau spatial transportant des robots autonomes se pose sur une planète inconnue, qui regorge de minerais rares à extraire. Une fois posé, le vaisseau déploie les robots sur la planète. Leur objectif est de trouver les minerais, de les récupérer, puis de les charger dans le vaisseau. Pour cela, ils peuvent communiquer entre eux afin de faciliter les recherches et d'optimiser leur travail. Une fois le vaisseau plein, ils doivent y retourner pour que celui-ci puisse décoller.

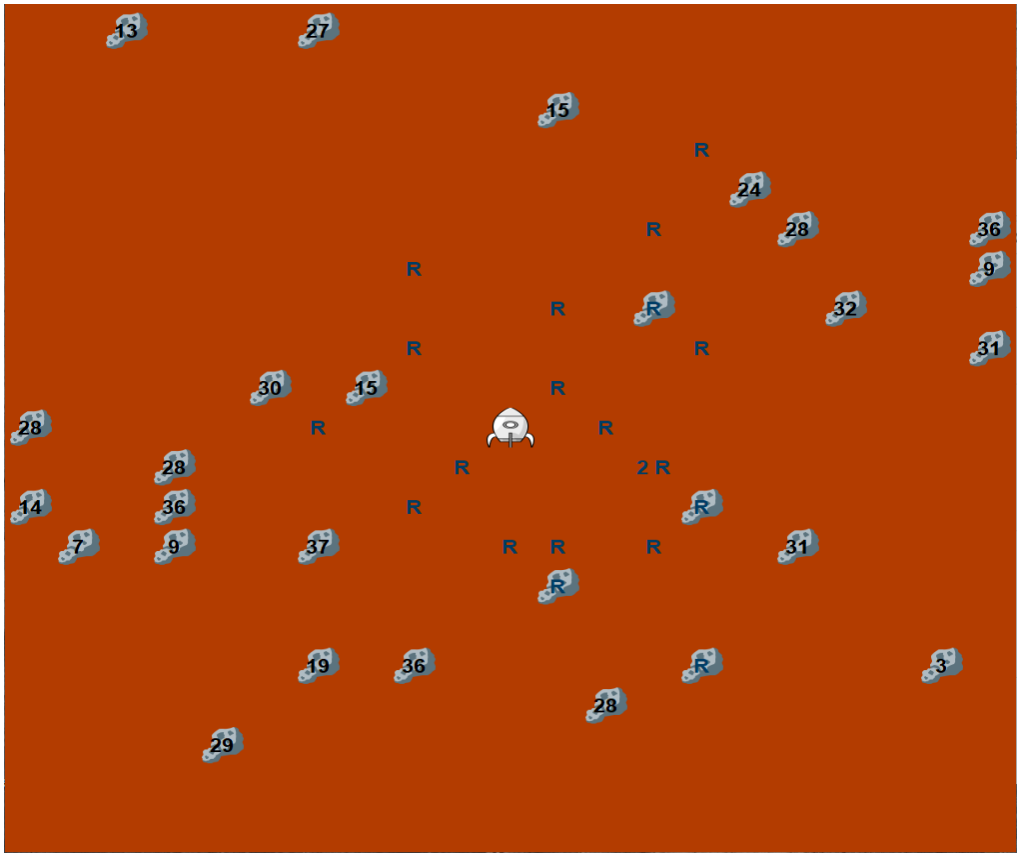

Figure 1. Screenshot de la simulation MinerAgent

La *figure 1.* illustre la simulation.

- Les robots sont représentés par la lettre "**R**". Si plusieurs robots se trouvent au même endroit, leur nombre est indiqué. (Eg. "2 R")
- Les minerais sont représentés par leur image ainsi qu'une indication sur leur quantité, affichée seulement quand aucun robot ne se trouve dessus.
- Le vaisseau spatial est représenté par une image.

Modélisation

Le système multi-agents *BDI* conçu pour ce projet est dit ***fermé*** (l'ensemble d'agents restant le même), ***homogène*** (les robots étant construits sur le même modèle) et ***non-mixte*** (aucun humain n'intervient dans la simulation). Cette partie décrit la modélisation et le fonctionnement de la simulation.

Planet

Avec *JADE*, toutes les entités sont des agents. Ainsi, la planète est donc aussi représentée par un agent *JADE*. La classe contient une grille avec la position et la quantité de tous les minerais, ainsi que la position de toutes les entités externes (vaisseau, robots). L'agent possède un seul *Behavior* (cyclique), qui consiste à répondre aux messages des autres agents. C'est de cette manière que la planète enregistre l'activité des entités externes sur celle-ci. À noter que chaque case représente une zone de la planète, d'où la possibilité d'être à plusieurs robots sur la même case (donc, dans la même zone). La taille de la zone est implicite et représente le champ de vision des robots.

Voici la liste des messages gérés :

- *PLANET_LAND* : un vaisseau se pose sur la planète. Réponse de consentement dans tous les cas.
- *SHIP_AGENT_DEPLOY* : un robot se déploie sur la planète depuis le vaisseau. Réponse de consentement dans tous les cas.
- *AGENT_MOVE* : un robot veut se déplacer dans une direction donnée. Réponse positive si il le peut, associée au nouvel environnement local du robot.
- *AGENT_MINE* : un robot veut miner à une position donnée. Réponse positive si possible, associée à la quantité minée. Sinon message de refus.
- *LEAVE_PLANET* : un vaisseau quitte la planète.

Ship

Un vaisseau possède (entre autres) une capacité de stockage maximale, une liste des robots dans le vaisseau (total), une liste des robots actifs (sur la planète) et un temps maximal d'opération. Quand il se pose sur la planète, il lui envoie un message afin de l'en informer. Une fois la réponse de celle-ci obtenue, il demande à tous les robots de se déployer. Le vaisseau à trois *Behaviors* :

- Timer (*TickerBehavior*) : déclenche la fin de l'opération et demande à tous les robots de revenir au vaisseau après un temps maximal atteint.
- CheckStatus (*Behavior*) : vérifie si la capacité maximale est atteinte. Déclenche la fin de l'opération si c'est le cas. Vérifie aussi que tous les agents sont rentrés avant de quitter la planète.
- HandleMessage (*CyclicBehavior*) : gère les messages reçus.

Voici la liste des messages gérés :

- *PLANET_LAND* : réponse (toujours positive) de la planète pour se poser. Demande ensuite aux robots de se déployer.
- *PLANET_DEPLOY* : confirmation de déploiement d'un robot. L'ajoute à la liste des robots actifs.
- *AGENT_DROP* : demande de déchargement de minerais d'un robot. Réponse informative avec le nombre de minerais ne pouvant pas être stockés.
- *AGENT_BACK* : un robot souhaite rentrer dans le vaisseau. Réponse de confirmation et l'enlève de la liste des robots actifs.

Nb : Le robot se pose généralement au centre de la planète, sauf s'il y a un dépôt. Dans ce cas, il se pose sur une zone aléatoire.

Robot

Un robot possède (entre autres) la liste des autres robots du vaisseau, sa position locale par rapport au vaisseau (*Belief*), une capacité de stockage maximale, la position associée à la quantité des dépôts découverts par rapport au vaisseau (*Belief*), une mémoire des zones précédemment explorées. Il existe deux types de robots : *FastRobot* et *BigRobot*. Le premier possède une capacité de stockage plus faible mais se déplace plus vite. Il cherche à exploiter les minerais les plus proches de sa position actuelle (*Intention plan*). Le deuxième possède une capacité de stockage plus élevée mais se déplace plus lentement. Il préfère s'orienter vers les minerais les plus gros en priorité (*Intention plan*). Afin de retrouver son chemin, le robot enregistre sa position locale par rapport au vaisseau (lieu de déploiement). En situation réelle, cette position pourrait être calculée à l'aide d'un vecteur directeur et d'une vitesse. La mémoire du robot lui permet de prioriser les déplacements vers des zones non visitées lors de sa phase d'exploration, afin d'optimiser les recherches. Il se déplace aléatoirement s'il a déjà visité toutes les zones autour de lui.

Un robot possède deux Behaviors :

- HandleMessage (*CyclicBehavior*) : gère les messages reçus.
- PlanetMining (*CyclicBehavior*) : comportement du robot. (*Desires* & *Intentions*)

Les *Desires* du robot sont de trouver les minerais, de les miner puis de les ramener au vaisseau. D'un point de vue global, les *Desires* du système sont de se poser sur la planète, remplir le vaisseau, puis de décoller. Le Behavior *PlanetMining* permet de choisir le *Desire* à l'instant *t* (*Active goal*) pour en faire une *Intention*.

La *figure 2.* décrit l'arbre de décision que suit un robot lorsqu'il est en opération sur la planète. Cet arbre de décision peut activer deux modes (*Intentions*) suivant le contexte actuel du robot : *mining* et *exploring*.

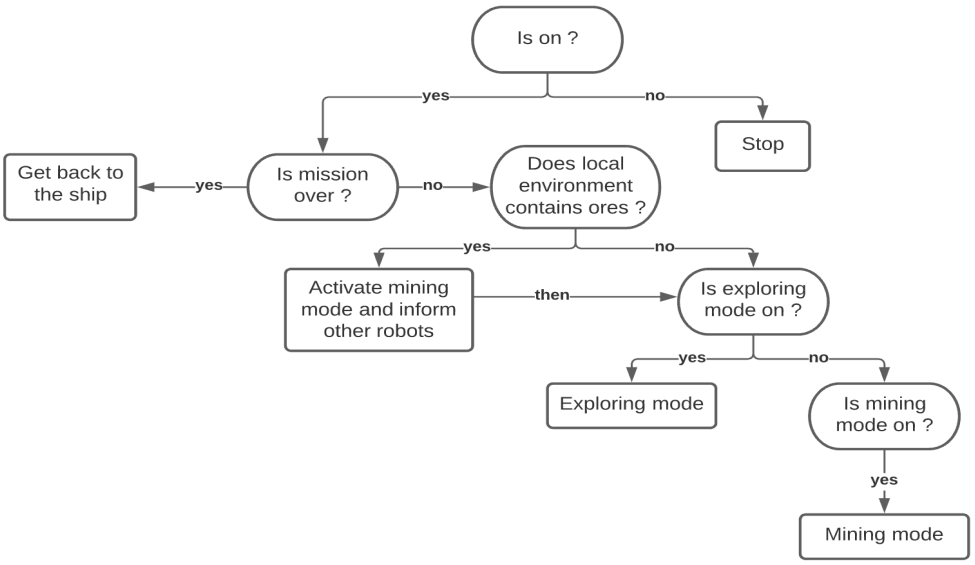

Figure 2. PlanetMining behavior decision tree

Le mode *exploring* vérifie s'il existe un dépôt actif (grâce aux messages reçus des autres robots). Si oui, le robot choisit le plus proche ou le plus gros selon sa stratégie et avance vers sa position. Sinon, il explore la planète en évitant de tourner en rond.

Le mode *mining* est décrit par l'arbre de décision *Figure* 3.

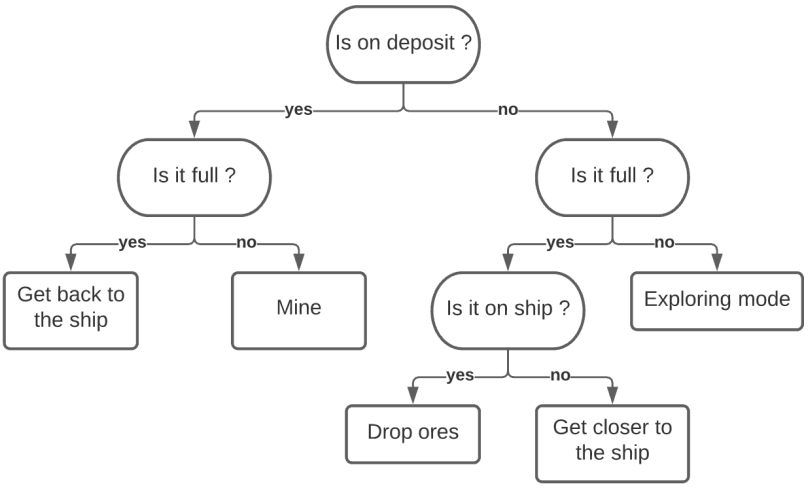

Figure 3. Mining mode decision tree

Voici la liste des messages gérés :

- *PLANET_DEPLOY* : le vaisseau demande au robot de se déployer sur la planète. Réponse de confirmation au vaisseau et informe la planète de son déploiement.
- *SHIP_AGENT_DEPLOY* : la planète accepte le déploiement. Déclenche le mode *exploring*.
- *AGENT_MOVE* : la planète confirme la demande de déplacement. Le robot sauvegarde son nouvel environnement local.

- *DEPOSIT_DISCOVERY* : reçoit la position d'un dépôt de minerais ainsi que sa quantité restante actuelle de la part d'un autre robot.
- *AGENT_DROP* : le vaisseau redonne les minerais qu'il ne peut pas stocker lors d'une demande de déchargement du robot. Déclenche le mode *exploring*.
- *AGENT_MINE* : la planète donne les minerais minés après une demande du robot. Si le robot n'a plus de stockage, les minerais en trop sont perdus. Si la planète refuse le minage, le robot actualise l'état du dépôt et informe les autres robots.
- *DEPOSIT_EMPTY* : informe les autres robots qu'un dépôt est vide. Le robot actualise aussi l'état du dépôt.
- *MISSION_OVER* : le vaisseau demande de terminer sa mission et de revenir.
- *AGENT_BACK* : le vaisseau accepte que le robot rentre dedans. Le robot informe la planète de son départ et passe en mode *off*.

GUI

Afin de visualiser la simulation, une interface graphique a été développée avec le framework *Swing* de Java. Lors de l'exécution, 2 fenêtres se présentent à l'utilisateur :

- Planet : vue omnisciente du dessus de la planète. Donne la position du vaisseau, des robots et des minerais avec leur quantité.
- Ship : panel d'information du vaisseau avec son stockage, le nombre de robots actifs, l'activité actuelle ainsi qu'un retour sur chaque agent (son statut, son mode activé, sa capacité et sa position locale).

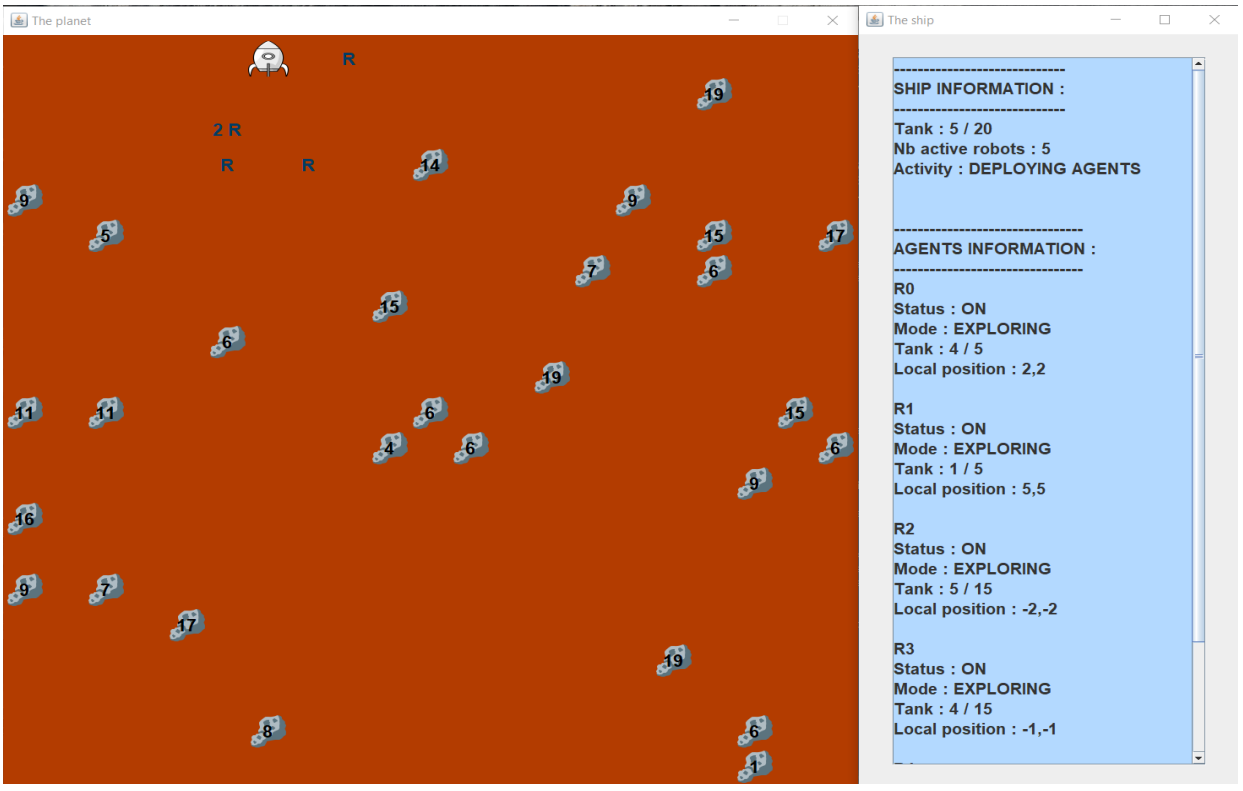

Figure 4. GUI de la simulation

Via la modélisation JADE, il est impossible de récupérer les informations des agents directement depuis un objet standard. Pour cela, il faut passer via une interface comme décrit *figure* 5.

```java
package ship;

public interface ShipInterface {
    void landOnPlanet();
    String getInfo();
    boolean onPlanet();
}
```

Figure 5. Code de l'interface pour récupérer les informations du vaisseau

Ainsi, chaque agent JADE implémente une interface permettant de récupérer les informations utiles à l'affichage. Il est ensuite possible de récupérer son instanciation via `ship.getO2AInterface(ShipInterface.class)` après l'avoir enregistré de cette manière `registerO2AInterface(ShipInterface.class, this)`.

À noter que dû au parallélisme de JADE (chaque agent étant un thread), l'interface graphique peut, dans certains cas, afficher un état t qui est déjà passé à $t+1$ dans la logique de la simulation. (Eg. robot affiché à une position mais qui est déjà à une nouvelle position). Cela ne change en rien au bon fonctionnement de la simulation.

Nb: JADE possède aussi une GUI pour visualiser et gérer les agents, ainsi que les échanges entre eux. Elle a été désactivée dans les scénarios pour ne pas encombrer l'écran de l'utilisateur.

Scénarios

Plusieurs scénarios ont été réalisés afin de tester la simulation :

- Scénario 1 : Vaisseau de capacité 30, temps maximal d'opération sur la planète de 2 minutes, composé de 5 robots (2 FastRobots, 3 BigRobots). La planète est de taille 13x13 et est composée de 20 minerais disposés aléatoirement avec une quantité maximale de 20.
- Scénario 2 : Vaisseau de capacité 120, temps maximal d'opération sur la planète de 2 minutes, composé de 20 robots (10 FastRobots, 10 BigRobots). La planète est de taille 21x21 et est composée de 40 minerais disposés aléatoirement avec une quantité maximale de 40.
- Scénario 3 : Le vaisseau demande aux robots de stopper l'opération après 10 secondes passées sur la planète.
- Scénario 4 et 5 : Scénario 1 avec que des FastRobots (4) et BigRobots (5)

Discussion

Pour conclure sur ce projet, on peut discuter de certains points d'implémentation pouvant être révisés.

Dans un premier temps, il est clair que le comportement global des robots peut être amélioré afin d'augmenter leur efficacité individuelle et de groupe (*Desires* & *Intentions*). Par exemple, il faudrait qu'ils fixent une limite de robots sur un dépôt afin de favoriser l'exploration de la planète. De plus, le vaisseau pourrait organiser les recherches en indiquant des directions à explorer aux robots, plutôt que d'avoir une exploration individuelle et partiellement aléatoire, pouvant rendre la recherche chaotique. Sinon, chaque robot pourrait décider lui-même d'une direction à explorer en s'organisant avec les autres. Pour finir, les robots pourraient communiquer entre eux et avec le vaisseau sur leur stockage actuel pour décider d'arrêter l'exploration, si le total de leur stockage permet de remplir le vaisseau (actuellement, les robots attendent d'être plein avant de revenir au vaisseau; l'utilité du BigRobot est donc moindre).

Dans un second temps, on pourrait aussi améliorer la modélisation de l'environnement afin de le rendre plus réaliste. Premièrement, la planète est représentée par une grille avec chaque case associée à une zone. On pourrait plutôt passer à une représentation avec des valeurs réelles (comme de réelles positions), ce qui demande tout de même de grosses adaptations. De plus, il faudrait ajouter la caractéristique *ronde* à la planète (n'en déplaise aux Platistes), pour que les robots puissent en faire le tour sans être bloqués par les bords de la grille. Finalement, on pourrait aussi intégrer des obstacles empêchant l'avancée des robots sur une position. On peut tout de même préciser que l'amélioration de l'environnement n'est pas l'objectif premier de ce projet, le but étant la conception d'un système multi-agents avec le framework JADE.

Pour conclure sur ce projet, la simulation multi-agents étudiée est très intéressante de par ses nombreuses améliorations possibles, rendant les agents de plus en plus intelligents et coopératifs entre eux. Par ailleurs, le framework JADE, en plus de faciliter les communications et le fonctionnement de la simulation, permet d'implémenter les comportements des agents de manière élégante. Ainsi, il devient plus simple de complexifier leurs agissements.