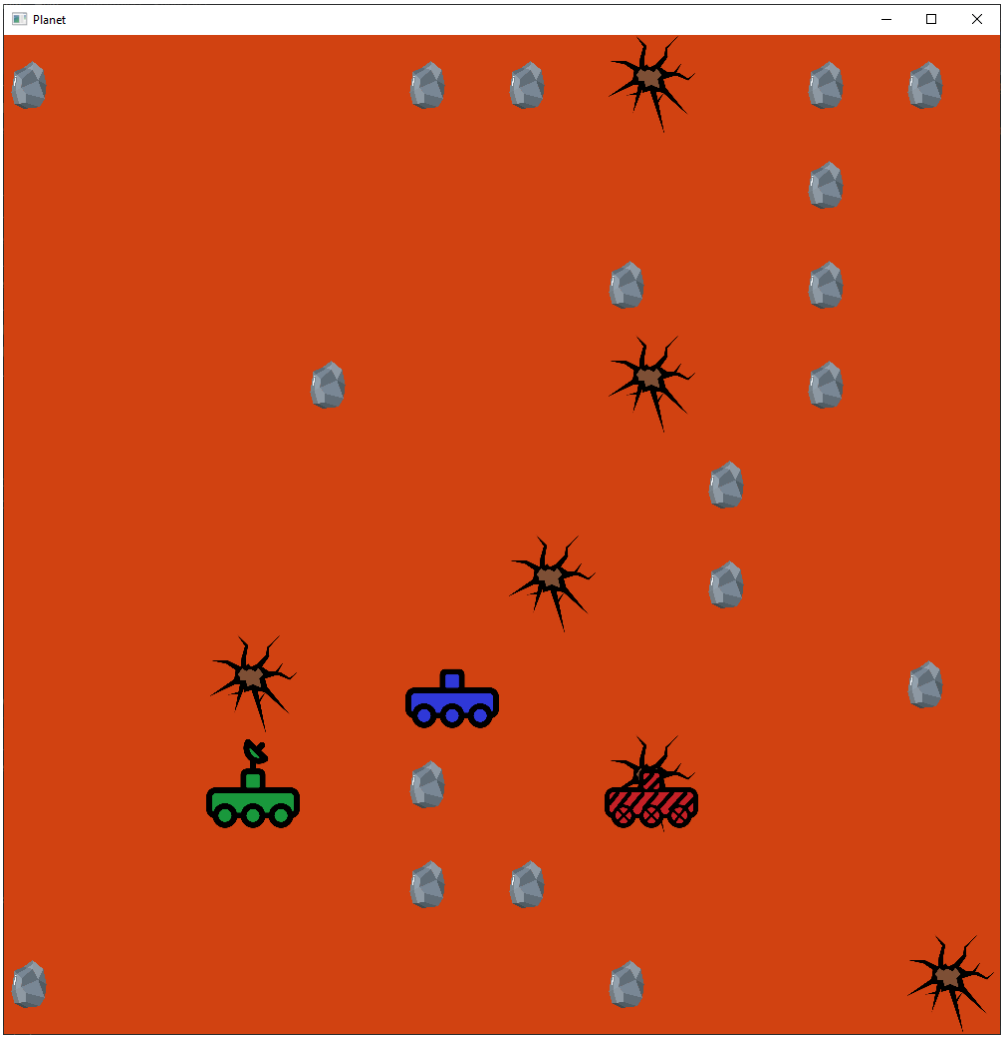
Rapport multi-agents and self systems

Projet Rover Martien

Michelland Yohan, Favre Victor



UNIVERSITÉ DE LYON
Lyon 1

I - Sujet choisi

Dans le cadre de l'UE multi-agents and self systems nous avons été amenés à réaliser un projet ayant pour but l'implémentation de l'architecture BDI pour des agents cognitifs. Pour ceci nous avons décidé de nous intéresser au cas d'étude ci-dessous.

Nous avons imaginé le cas d'un ensemble de rovers autonomes sur une planète inconnue. Ces rovers ont pour objectif l'analyse d'échantillons présents à certains endroits de la planète. Pour atteindre leur objectif, ceux-ci doivent chercher l'emplacement des échantillons pour ensuite les ramasser et les analyser. Le tout en évitant les cratères et en gérant leur niveau de batterie. Ils ont aussi comme objectif d'aider les autres rovers dans le cas ou l'un deux serait bloqué après être tombé dans un cratère. Les rovers sont donc collaboration.

Nous avons implémenté notre projet en Java, en basant notre système multi-agents sur le framework Jade. L'interface graphique est codée à l'aide de la bibliothèque JavaFX.

II - Approche voyelle

L'approche voyelle nous a guidé lors des premières étapes de la conception. Dans cette partie nous allons présenter le modèle que nous avons choisi.

E - Environnement

Il s'agit d'une grille carré, chaque case est ou bien vide, ou bien un échantillon collectable, ou bien un cratère à éviter. La grille est générée aléatoirement à chaque simulation.
Un système de cycle jour/nuit a été mis en place : A chaque itération du monde, une heure passe, jusqu'à 24. De 18 à 24, il fait nuit.
A noter que la Planete (qui contient l'environnement) est également un agent au sens de Jade, afin de permettre les échanges de messages avec les rovers.

A - Agent

L'agent est un rover capable de se déplacer horizontalement ou verticalement de 1 case dans l'environnement à chaque action. Chaque action qu'il effectue lui fait perdre de la batterie. Plutôt que se déplacer, il peut également :

- Récolter un échantillon présent sur sa position
- Analyser 4 échantillons déjà récoltés pour créer un rapport d'analyse.
- Recharger sa batterie lorsqu'il fait jour
- Se casser lorsqu'il tombe sur un cratère
- Réparer un autre rover cassé si il est à une case de sa position

En terme de perception, l'agent n'a accès qu'à sa case actuelle (à cause de l'épais brouillard sur Mars), à l'heure qu'il est (informations transmises par l'environnement) et aux informations transmises par les autres agents.
Les motivations et les connaissances de l'agent seront détaillées ci-après dans la partie **Modélisation BDI.**

I - Interactions

Comme l'entend le framework Jade, les interactions sont toutes effectuées par échange de messages, notamment les rovers pour communiquer entre eux. Voici les types de messages transitant entre les agents au cours de la simulation.

- “Mayday-5,6” : l’agent avertit tout le monde qu’il est cassé, et qu’il a besoin d’une assistance. Dans le même temps, il informe tout le monde de sa position (5,6).
- “Helping” : l’agent informe un agent en difficulté qu’il est en route pour l’aider.
- “NoMayday” : l’agent avertit tout le monde qu’il n’a plus besoin d’assistance car un agent a déjà proposé son aide.
- “Save” : l’agent est à côté de l’agent cassé, il le répare, ce qui lui permet de reprendre son exploration

De même, les interactions de l’agent avec l ‘environnement sont envoyées sous forme de message à l’agent Planet, qui modifie alors l’environnement en conséquence. (Nouvelle position de l’agent, nouveau statut de l’agent, échantillons récemment récoltés).

O - Organisation

Deux types d’agents interagissent :

- Planet est l’agent unique représentant l’environnement. Il est responsable de prodiguer une perception à chaque Rover et chaque itération. Il doit également réceptionner les messages de l’ensemble des Rovers afin de mettre l’environnement à jour.
- Les Rovers sont l’autre type d’agents, ils peuvent être deux, trois ou beaucoup plus. A part leur position initiale dans l’environnement qui est aléatoire, il sont initialisés avec les mêmes paramètres et sont hiérarchiquement égaux.
  Les agents individuels ayant des buts compatibles, ils sont en collaboration. Leur compétences individuelles sont insuffisantes (car il ne peuvent pas se réparer tout seul), mais les ressources sont suffisantes. Il s’agit donc d’une collaboration simple.

II - Modélisation BDI

Afin d'obtenir le comportement souhaité pour nos agents, nous avons décidé de modéliser les différentes composantes du BDI de la façon suivante.

A. Beliefs (classe *bdi.Beliefs*)

Selon les principes BDI les belief contiennent les informations que notre rover possède. Ce sont des informations subjectives dans le sens ou elles sont propres à l’agent et peuvent être fausses. Il y a donc un caractère d’incertitude. Voici quelques exemples de beliefs pour le rover :

- Sa position, en x et y

- La position d'un rover en difficulté
- Sa batterie restante
- Le temps restant avant le coucher de soleil
- La liste des cratères déjà découvert
- La liste des cases déjà visitées

B. Désires (classe *bdi.Desire*)

Les désirs de l'agent représentent les états de monde ou de lui-même qu'il souhaite voir accomplis. Le Rover a deux objectifs : premièrement il souhaite analyser un maximum d'échantillons (Progress). Deuxièmement, il souhaite que les autres rovers soient en état de fonctionner (Help).

Ces désires étant assez généraux, il est nécessaire d'avoir des intentions servant à représenter les différentes actions qu'entreprend l'agent pour accomplir ces désirs. Cette répartition étant exprimée ci-dessous.

C. Intentions (classe *bdi.Intention*)

Les intentions de notre rover représentent les sous actions nécessaires à la réalisation de ses désirs. La composante *Decision Process* (classe *bdi.Decision_Process*) permet à chaque itération de choisir une intention à partir des Beliefs, Desires, et Intentions de l'agent, en suivant un ordre de priorité. La liste des intentions et de leur conditions de réalisation étant:

Dans le cas du désir Progress:

- Analysing : Si il a accumulé assez d'échantillons, il décide en priorité de les analyser
- Gathering: Si un échantillon est présent, il décide de le ramasser
- Exploring : Par défaut, il décide de se déplacer vers une case voisine, en prenant en compte la liste des cases déjà visitées et la liste des cratères connus.

Dans le cas du désire Help:

- Reaching : il décide de se diriger vers le rover ayant besoin d'assistance, en ignorant les échantillons qu'il croise.
- Saving : il décide de réparer le rover endommagé sur un cratère pour de lui.

Enfin, certaines intentions sont communes aux deux désires :

- Mayday : S'il s'est endommagé sur un cratère, il décide d'envoyer un message d'appel à l'aide aux autres rovers

- Recharging : Si sa batterie est trop faible ou que la nuit approche, il décide de la recharger.

III Implementations

Il sera décrit dans cette partie comment a été implémenté l'architecture BDI.

A. Algorithme

Dans notre simulation, l'algorithme de contrôle BDI est appelé à chaque itération pour faire évoluer les désirs et les intentions de nos agents :

obtenir nouvelles perceptions **p**
B = ***revc***(**B**, **p**)
I = ***options***(**D**, **I**)
D = ***des***(**B**, **D** ,**I**)
I = ***filtre***(**B**, **D**, **I**)
PE = ***plan***(**B**, **I**)
exécuter(**PE)**

B : beliefs , **I** : intentions, **D** : desire

- ***revc*** **:** fonction de révision des croyances de l'agent lorsqu'il reçoit de nouvelles perceptions sur l'environnement, où **P** représente l'ensemble des perceptions de l'agent. (Fonction *perception().)*
- ***options*** **:** fonction qui représente le processus de décision de l'agent prenant en compte ses désirs et ses intentions courantes. (Fonction *Decison_Process.option(D,I, rover))*
- ***des*** **:** fonction qui peut changer les désirs d'un agent si ses croyances ou intentions changent, pour maintenir la consistance des désirs de l'agent (on suppose dans notre modèle que l'agent a toujours des désirs consistants). (Fonction *Decison_Process.des(B,D,rover)*)
- ***filtre*** **:** fonction la plus importante car elle décide des intentions à poursuivre.(Fonction *Filter.filter(D,I,rover)*)

B. Cognition

Dans le cadre de l'approche cognitive nous cherchons à obtenir des comportements individuels intelligents. Dans ce but nous avons implémenté different comportements comme :

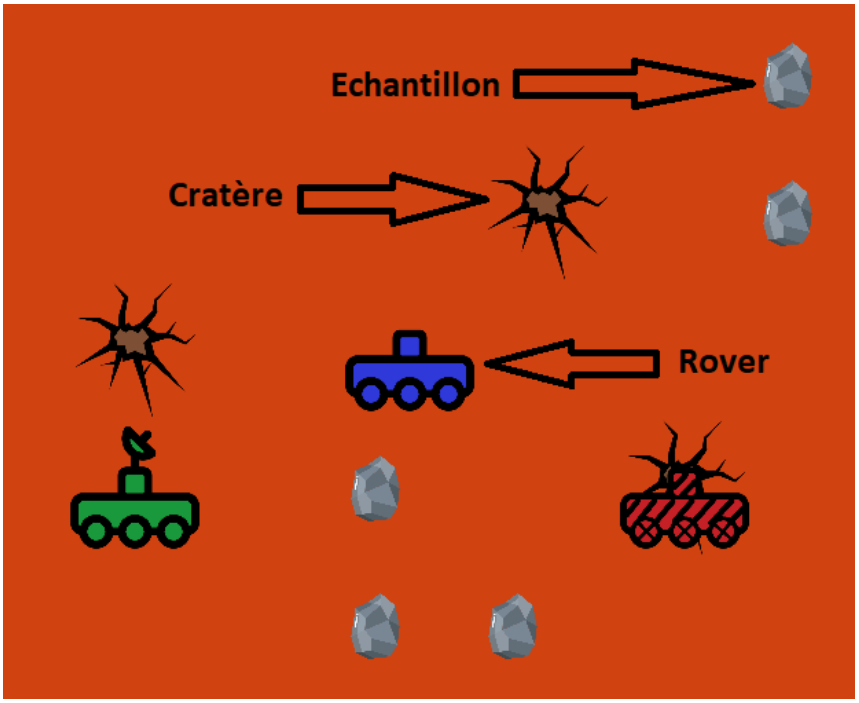
- Un historique des cases dans lesquels sont tombés les rovers, pour éviter de se déplacer sur une case contenant un cratère. Mis à jour lorsqu'un rover reçoit un

message d'appel à l'aide ("mayday-5,6"), car il en déduit qu'un cratère est sur cette position.

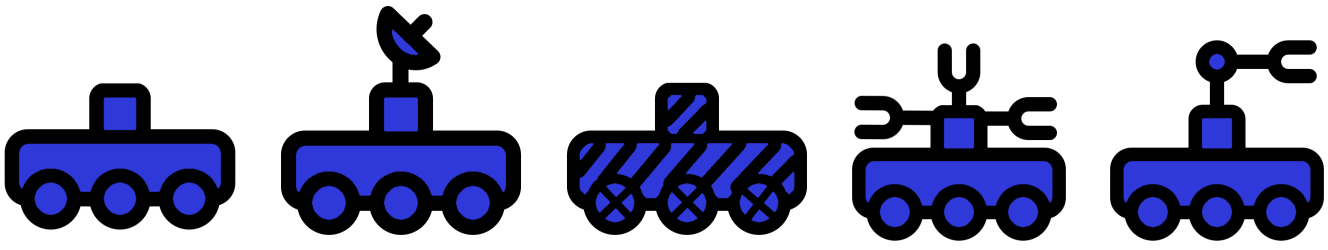
- Un historique des cases déjà visitées pour améliorer l'exploration. Un agent préfère visiter une nouvelle case dans la mesure du possible.
- Une recharge de batterie préventive pour avoir de la batterie durant la nuit. L'agent calcule le temps restant avant la nuit, ainsi que sa batterie restante, et en déduit s'il est important de recharger ou pas.

IV Application

La classe *application.App* observe l'agent Planet afin de représenter l'environnement et les agents.
Chaque Rover possède une couleur afin de l'identifier, et plusieurs formes pour connaître son intention actuelle.



Les différente etats du Rover sont représenté par :



Exploring
Reaching
Gathering

Recharging

Mayday

Analysing

Saving

V - Conclusion

Ce projet a été très intéressant à implémenter. Le lien entre Jade et l'interface graphique a suscité quelques difficultés de conception, mais une fois en place nous avons pu nous concentrer sur l'architecture BDI en utilisant l'interface pour debugger. L'approche BDI nous a convaincu par sa scalabilité : il est très facile d'ajouter une nouvelle Intention et de nouveaux Behavior.
De plus, nous avons pu remarquer la puissance des différents types de Behavior de Jade, en particulier les FSM Behavior, qui malheureusement, ne se prêtaient pas particulièrement à notre conception du projet.
Pour ce qui est des améliorations possibles :

- Les échanges de messages pourraient être mieux gérés à l'aide d'ontologies Jade, non utilisées suite à une découverte tardive.
- Au niveau de l'interface, plusieurs informations sont manquantes et pourraient être ajoutées. (niveau de batterie, nombre d'échantillons analysés).