Jocelyn BOURNIER
Maeva SOMNY

M2 IA
2023/2024

TP Jade
Agents missionnaires

I. Modélisation des agents

Pour commencer, nous allons vous présenter la modélisation de nos agents ainsi que leurs comportements. Basiquement, l'agent va parcourir l'environnement, et miner les tas de minerais qu'il trouve, tout en veillant à la place restante dans son sac et à son niveau de carburant.

Le comportement `Wandering` permet à l'agent de se déplacer et de miner tant qu'il reste suffisamment de place dans son sac et un niveau de carburant suffisant pour rentrer à la base.

Pour vérifier que le niveau de carburant est suffisant pour rentrer à la base, l'agent va construire **un graphe** représentant l'environnement au fur et à mesure qu'il le parcourt. Chaque nœud du graphe correspond à une "case" de l'environnement, et les arêtes, les liens que l'agent à découvert entre ces cases. Grâce à ce graphe, l'agent est capable de trouver le plus court chemin entre sa position et celle de base. La taille de ce chemin multipliée par le coût d'un déplacement permet donc à l'agent de s'assurer qu'il a toujours suffisamment de carburant pour rentrer. Afin de faire en sorte que l'agent ait une marge de sécurité, pour éviter un autre agent par exemple, nous avons ajouté dans notre calcul une **hystérésis** de (coût déplacement * 3). Ainsi, l'agent s'assure de toujours avoir plus de carburant que *((taille chemin vers base * coût déplacement) + hysteresis)*, sinon il rentre.

La fonction `moveToDirection` permet à l'agent de se déplacer dans l'environnement (en prenant en compte les limites de celui-ci). La fonction `checkOres()` permet à l'agent de vérifier le nombre de minerais que contient la "case" de l'environnement sur laquelle il se situe. Si la case ne contient pas de minerais, l'agent va boucler sur le comportement `Wandering` jusqu'à ce que ce ne soit plus possible.

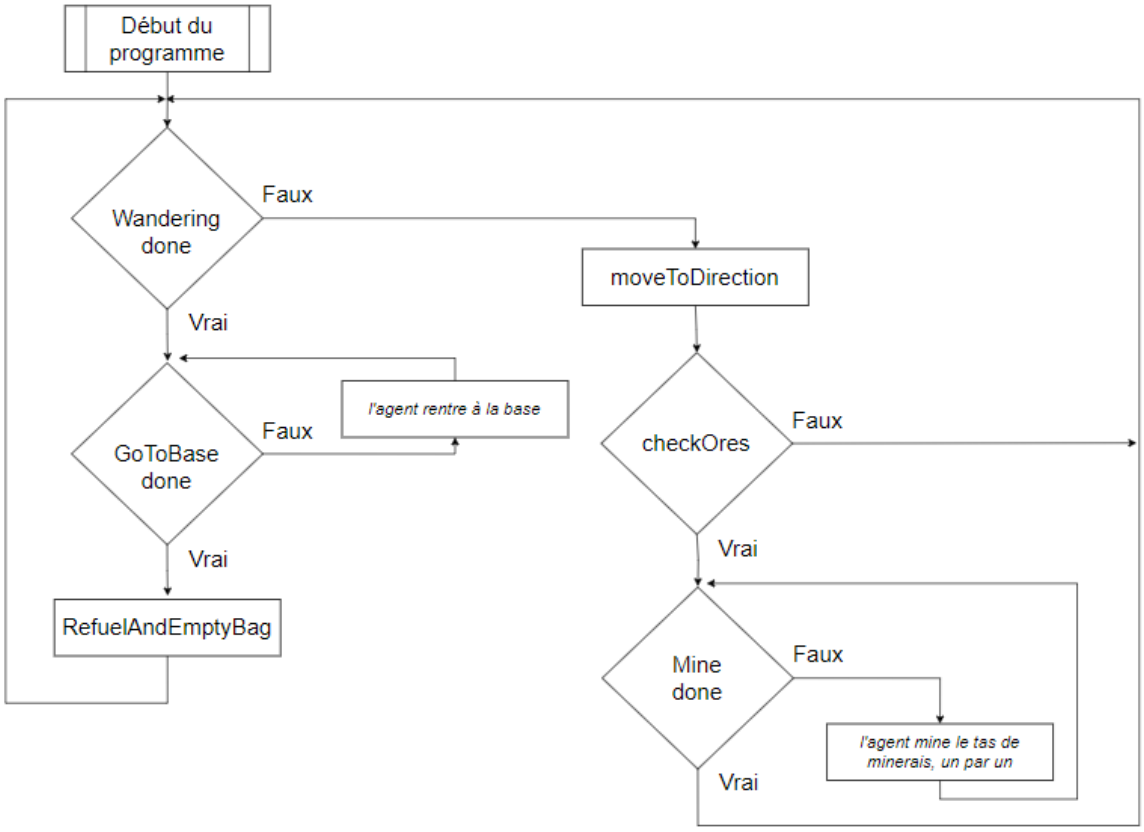
Dans l'autre cas, c'est le comportement `Mine` qui prend le relais, permettant à l'agent de miner le tas de minerais jusqu'à ce qu'il reste suffisamment de place dans son sac et un niveau de carburant suffisant pour rentrer à la base.

Le comportement `GoToBase` permet à l'agent de se déplacer pas à pas jusqu'à la base. Une fois arrivé (quand sa position est égale à celle de la base), il va déclencher le comportement `RefuelAndEmptyBag` qui permet de remplir son carburant et de vider son sac.

En bref, notre agent adopte le comportement décrit dans l'algorigramme en page suivante.

En d'autres termes, tant qu'il le peut, l'agent va se déplacer, cartographiant l'environnement dans lequel il évolue sous forme d'un graphe. A chaque déplacement, l'agent va vérifier s'il peut miner (en cas de présence d'un tas de minerais) ou doit rentrer à la base, en fonction de contraintes liées au carburant et à la place disponible dans son sac. Tant que ces contraintes ne sont pas atteintes, l'agent va se déplacer et miner.

Cependant, lorsque l'agent est sous l'une de ces contraintes, il va rentrer à la base afin de vider son sac et remplir son carburant. A la suite de ça, il peut reprendre son cheminement comme décrit au paragraphe précédent.

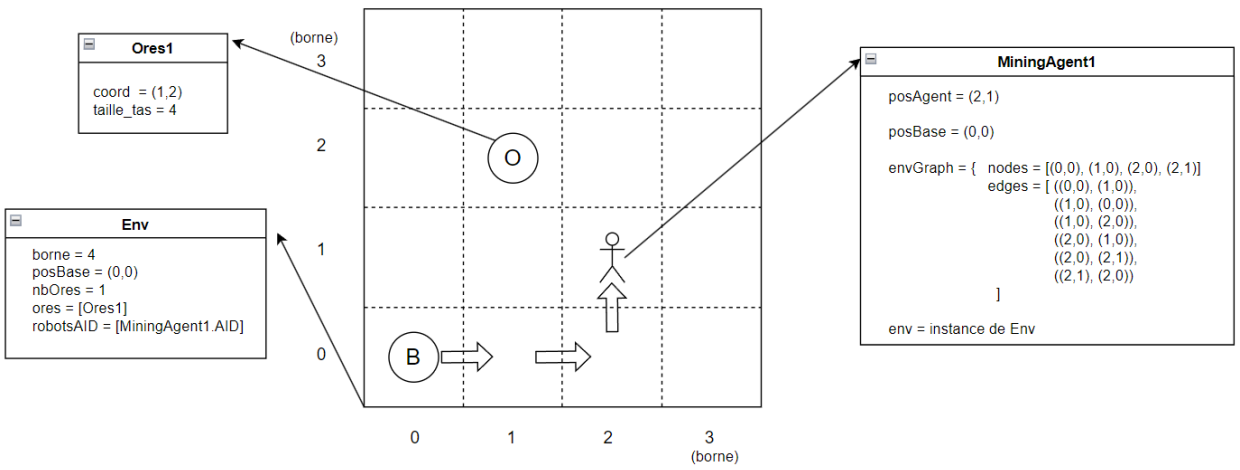


II. Modélisation de l'environnement

L'environnement quant à lui est modélisé par la classe **`Env`**, contenant la borne supérieure de l'environnement (`borne`, on considère alors que l'environnement est carré), les coordonnées de la base (`posBase`), ainsi qu'une liste contenant les tas de minerais (`ores`).

Afin de modéliser au mieux cet environnement (et le graphe que l'agent le créer afin de le modéliser), nous avons créé une classe **`Coord`** afin de stocker nos coordonnées (positions des différents agents, de la base, des minerais, et noeuds), ainsi que la classe **`Ores`**, permet de fournir les informations d'un tas de minerais (ses coordonnées, sa taille).

Ainsi, notre environnement peut être vu comme :



Pour nos tests, nous avons mis en place un environnement de 10x10, avec 5 tas de minerais de taille comprise entre 2 et 4, positionnés de manière aléatoire (en veillant cependant à ce qu'un minerais ne soit pas positionné sur la base et que deux minerais ne soient pas sur la même case).

III. Interactions des agents

Une fois notre modélisation de l'environnement et de notre agent testée avec un agent, nous avons cherché à ajouter la possibilité d'avoir plusieurs agents dans un même environnement. Après quelques modifications dans notre implémentation nous avons ajouté de l'interaction entre nos agents.

Ces deux agents ont un comportement identique, tel que décrit en partie 1. Toutefois, les agents communiquent leurs déplacements aux autres agents, dans le but de ne pas entrer en collision.

À chaque déplacement, chacun des deux robots envoie sa nouvelle position à l'autre robot. Ce dernier garde en mémoire cette position et, lors de l'évaluation des mouvements possibles, on vérifie si l'un des mouvements générera une collision, et si oui, on le retire de la liste.

Nous n'avons pas eu le temps de créer de comportement gérant les informations récoltés concernant les positions des minerais, mais les robots s'échangent aussi des messages pour s'informer des localisations connues des minerais. Dans le cas où un agent part d'une case à laquelle des minerais sont encore présents, il envoie un message pour le signaler à l'autre robot, avec le nombre de minerais restants.

IV. Scénarios

Le scénario classique que nous avons utilisé tout au long de notre phase de test et de développement et donc celui déclenché par le code disponible sur le git est le suivant :

- Dans un environnement de taille 10x10 contenant 5 tas de minerais de taille variant entre 20 et 40, deux agents apparaissent dans une base située aux coordonnées (0,0).

- Ces deux agents ont une quantité initiale de carburant de 100, ainsi qu'une taille de sac de 5.

- Les Agents vont explorer l'environnement, en créant chacun un graphe permettant de cartographier l'environnement, cela leur permettant de se déplacer de manière efficace dans celui-ci, et de savoir lorsqu'ils doivent rentrer à la base pour se recharger en carburant.

- Lorsque deux agents se déplacent, ceux-ci communiquent leur position aux autres agents afin de faire en sorte que ceux-ci n'entrent pas en collision. Nous avons en effet choisi de faire en sorte qu'une "case" de l'environnement ne puisse contenir qu'un seul agent à la fois.

V. Améliorations possibles

Nous n'avons malheureusement pas pu réaliser l'ensemble des fonctionnalités auxquelles nous avons pensé. En effet, nous avons eu quelques difficultés à adapter notre implémentation à la présence de plusieurs agents, ce qui nous a retardés.

Cependant, nous avions pensé aux fonctionnalités suivantes :

- Les agents ne gardent pas de trace des minerais encore non vides. Il serait intéressant de faire en sorte que les agents marquent les positions des minerais non vides afin d'y retourner une fois leur sac vidé et/ou leur carburant rempli.
- Les agents prennent en compte les informations des autres agents afin de se diriger vers les minerais trouvés. Cela serait très intéressant dans notre cas puisque les agents ont une cartographie de l'environnement ce qui leur permettrait d'optimiser leurs déplacement entre leur position et la position du minerais partagé par un autre agent.
- Les agents s'échangent les nœuds et arêtes de leurs graphes dans le but de cartographier plus vite l'ensemble de l'environnement. Cela permettrait aux agents d'optimiser leurs trajets puisqu'ils découvrent plus vite tous les chemins possibles.
- Après être rentrés à la base bredouille (c'est à dire sans minerais, par la contrainte du carburant), les agents partent en quête d'une cartographie complète de l'environnement, dans le but de s'assurer que chacune des "cases" à été visitée, et de s'assurer que celles-ci ne contiennent plus aucun minerais. Dans le cas où les agents ont confirmé le fait que l'environnement ne contient plus aucun minerais, ils peuvent le communiquer à la base, qui, dans un scénario hypothétique, les rappatrieraient pour les conduire vers un nouvel environnement pleins de potentielles ressources.
- Il pourrait être intéressant d'ajouter un agent "citerne", qui est capable de remplir en carburant les autres agents. Ainsi, lorsqu'un agent n'a plus assez de carburant pour continuer à explorer/miner, mais qu'il lui reste de la place dans son sac, il pourrait demander à l'agent "citerne" de venir le remplir afin d'éviter de rentrer à la base avant de reprendre son minage.