Université Cheikh Anta Diop

Ecole Supérieure Polytechnique de Dakar

Systèmes Multi Agents
Etude de cas

2

Auteurs :
Papa Latyr Mbodj
Cheikh Ahmet Tidjane Sankare

Superviseur :
Pr Alassane Bah

18 janvier 2018

Introduction

Afin d'appliquer les notions enseignées en Cours de Systèmes Multi Agents, nous avons eu à réaliser une étude de cas avec GAMA,un environnement de développement orienté modélisation et simulation de systèmes à base d'agents. Cet étude de cas concerne une ***exploitation d'un territoire minier par une population de robots***. Les robots doivent trouver tous les minerais présents dans le territoire et les amener à une base.
L'étude de cas est décliné en deux versions : une première version avec des robots réactifs et une deuxième version avec des robots cognitifs. Nous avons structuré notre rapport de la façon suivante : Tout d'abord, dans une première partie nous faisons une spécification générale dans laquelle nous fixons les définitions de chacun des éléments manipulés dans notre étude de cas. Puis dans une seconde partie, nous présentons la première version du robot, le robot réactif. Ensuite, dans une troisième partie, nous déclinerons le robot cognitif. Puis nous ferons des test de sensibilité et comparaisons des deux versions avant de finir par une conclusion.

Table des matières

Chapitre 1

Spécifications

1.1 Environnement

Notre environnement est une grille carrée de 20 lignes et 20 colonnes donc 400 cellules de type "test_grid". A partir de chaque cellule, il est possible d'obtenir la liste de ses 4 voisins les plus proches ; celui de gauche, celui de droite, celui d'en haut et celui d'en bas. Toutes les interactions se font dans cet environnement.

```
grid test_grid width: 20 height: 20 neighbours: 4 {
    bool est_mine <- false;
    rgb color <- #white;
    list<test_grid> neighbours <- self neighbours_at 1;
}
```

FIGURE 1.1 – Définition d'une grille de l'environnement

1.2 Minerais

Pour représenter les minerais dans l'environnement, nous prenons aléatoirement autant de cellules que de minerais que nous voulons représenter. Afin de faire de ces cellules des mines, nous leur attribuons un état miné pour de dire qu'elles contiennent des mines. Aussi, nous colorons les cellules minées en noir.

```
int nbMines <- 0;
/* On parseme les minerais dans l espace  */
loop while: (nbMines < nbMinerais){
    test_grid cellule <- one_of(test_grid) ;
    if(cellule != test_grid grid_at {0,0}){
        if(!cellule.est_mine){
            cellule.est_mine <- true;
            cellule.color <- #black;
            nbMines <- nbMines +1;
        }
    }
}
```

FIGURE 1.2 – Dissémination des minerais

1.3 Base

Elle représente l'agent passif qu'est la base ici, c'est a dire qu'elle ne fait rien et ne sert ici que de cible de dépôt pour les minerais collectés. Elle sauvegarde la référence vers la cellule de grille dans laquelle elle se trouve.

```
species base_species {
    init{
        name <- "Base";
        shape <- square(5);
        test_grid cellule_base <- test_grid grid_at {0,0};
        location <- cellule_base.location;
    }
    list<test_species> test_species_inside_base -> {test_species inside self};
    aspect standard_aspect{
        draw square(5) color: #orange;
    }
}
```

FIGURE 1.3 – Base de dépôt des minerais

1.4 Robot réactif

C'est notre type de robot de base. Il représente des robots réactifs, c'est a dire qui entreprennent des actions qu'en fonction de leur environnement sans aucun apprentissage. Ils n'ont aucune possibilité de connaitre à l'avance la localisation de la base de dépôt des minerais. Pour trouver la base, ils se déplacent aléatoirement de voisins en voisins jusqu'à'à se trouver à la base Le robot a un nom, sa position actuelle dans l'environnement ainsi que le minerai qu'il porte (potentiellement). Il a aussi un objectif qui peut être soit de chercher des minerais, soit de prendre un minerai, soit d'aller a la base, soit de déposer un minerai a la base.
Lorsqu'on l'introduit dans l'environnement lors de l'initialisation, son objectif est de chercher un minerai. Des qu'il en trouve, son objectif devient d'aller a la base. Une fois a la base, il doit y déposer le minerai. Ses objectifs sont atteints à l'aide de réflexes [1].

Dans le cas de notre robot réactif, les réflexes définis sont les suivants :
— **chercher_minerai** qui permet au robot de rechercher des minerais dans l'environnement ;
— **prendre** qui permet au robot de prendre un minerai trouvé ;
— **aller_base** qui permet au robot d'aller à la base ;
— **deposer** qui permet au robot de déposer un minerai dans la base.

```
species test_species{
    test_grid minerai_actuel;
    test_grid position_actuelle;
    string objectif <- "chercher" among: ["chercher","prendre","aller_base","deposer"];
    reflex chercher when: (objectif = "chercher") {
    reflex prendre when: (objectif="prendre") {
    reflex aller_base when: (objectif="aller_base") {
    reflex deposer when: (objectif="deposer") {
    reflex arreter_recherche when: (nbMinesDeposes = nbMinerais){
    action depot{
    init {
    aspect standard_aspect{
}
```

FIGURE 1.4 – Robot réactif

1. Les réflexes sont des ensembles d'instructions qui s'exécutent après chaque cycle du robot (pas de temps). Elles représentent ainsi les compétences spécifiques des agents et correspondent aux opérations d'une classe ou aux activités dans un diagramme d'activité.

1.5 Robot cognitif

C'est notre deuxième type de robot. Il s'agit d'une version améliorée du robot réactif. Ce robot a les mêmes réflexes que le premier type de robot. La différence se trouve au niveau du réflexe aller_base qui est faite ici de façon plus intelligente. En effet, tout comme le robot réactif, le robot cognitif n'a aucune connaisance de l'emplacement de la base à l'avance. Mais, une fois après y être allé, il mémorise cet emplacement. Cela fait qu'après, il ne recherche plus la base mais y va directement en prenant le chemin le plus court.

```
species smart_species {
    file my_icon <- file("../images/robot.png") ;
    float size <- 3.0;
    test_grid position_actuelle;
    test_grid minerai_actuel;
    bool premiere_fois <- true;
    base_species la_base;
    string objectif <- "chercher" among: ["chercher","prendre","aller_base","deposer"];
    reflex chercher when: (objectif = "chercher") {
    reflex prendre when: (objectif="prendre") {
    reflex aller_base when: (objectif="aller_base") {
    reflex deposer when: (objectif="deposer") {
    reflex arreter_recherche when: (nbMinesDeposes = nbMinerais){
    action depot{
    init {
    aspect standard_aspect{
    aspect icon {
}
```

FIGURE 1.5 – Robot cognitif

Chapitre 2

Première Version : Robot Réactif

2.1 Diagramme de Classes

Le diagramme de classe est censé représenté la structure statique d'un système. Seulement comme dans GAMA tout est agent, les classes que nous manipulerons ici ne représentent en fait que des agent formalisés selon UML. Elles dérivent donc toutes de la classe "agent" nativement définie dans la documentation de GAMA. Nous ne représenterons pas cette classe dans nos diagrammes ici. Ci-dessous le diagramme de classe du modèle avec Robots Réactifs.

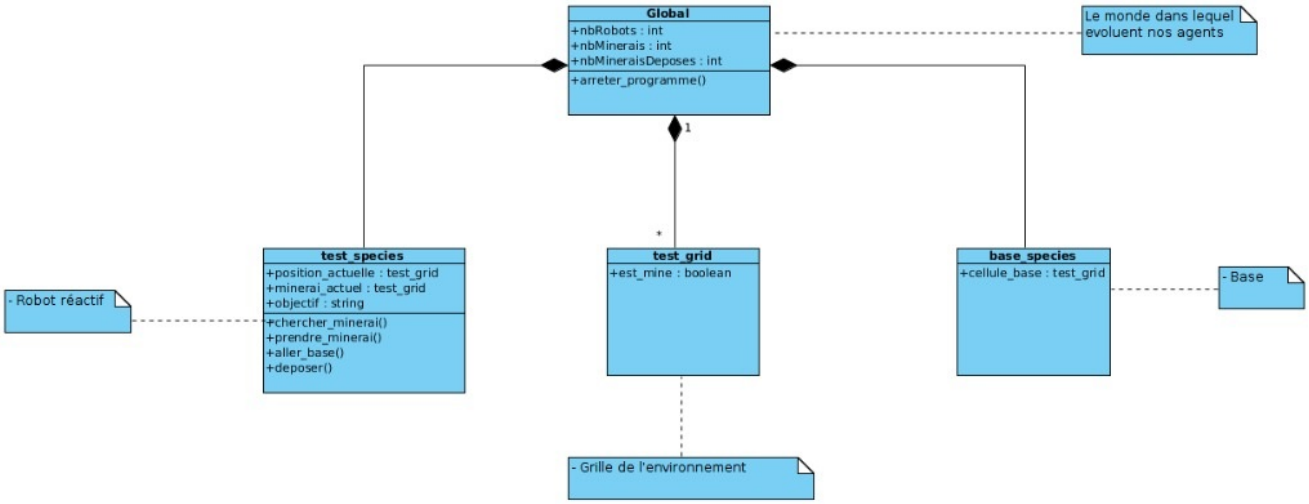


FIGURE 2.1 – Diagramme de classe du robot réactif

Dans ce diagramme nous avons :

Le Monde "global" Il représente l'environnement dans lequel s'exécute la simulation. Il est lui même un agent et sert de conteneur a tous les autres agents du modèle, ce qui justifie la relation de composition qu'il partage avec toutes les autres classes. Comme son nom l'indique, il contient l'ensemble des variables et configurations globales du modèle et des expérimentations que nous ferons.

La grille (les Cellules) "test_grid" Ce sont les cellules de la grille dans laquelle notre simulation se déroule. Chacune d'entre elle est dotée d'un attribut additionnel "est_mine" qui permet de connaitre si la zone recouverte par la cellule contient ou non un minerai.

la Base "base_species" c'est l'agent qui représente la base de stockage des minerais. Il est associé à une des cellules de la grille choisie arbitrairement (ici cellule de la première ligne et de la premiere colonne) avec qui elle partage ainsi la même position.

Les Robots Réactifs "test_species" C'est notre Robot collecteur, il est réactif mais pas "intelligent" puisqu'il n'a aucun moyen de savoir à priori où se trouve la base. Tout ce qu'il sait faire c'est se déplacer aléatoirement de voisins en voisins jusqu'à ce qu'il trouve la base dans sa zone de perception (cellule actuelle + cellules voisines).

2.2 Diagramme d'Activités

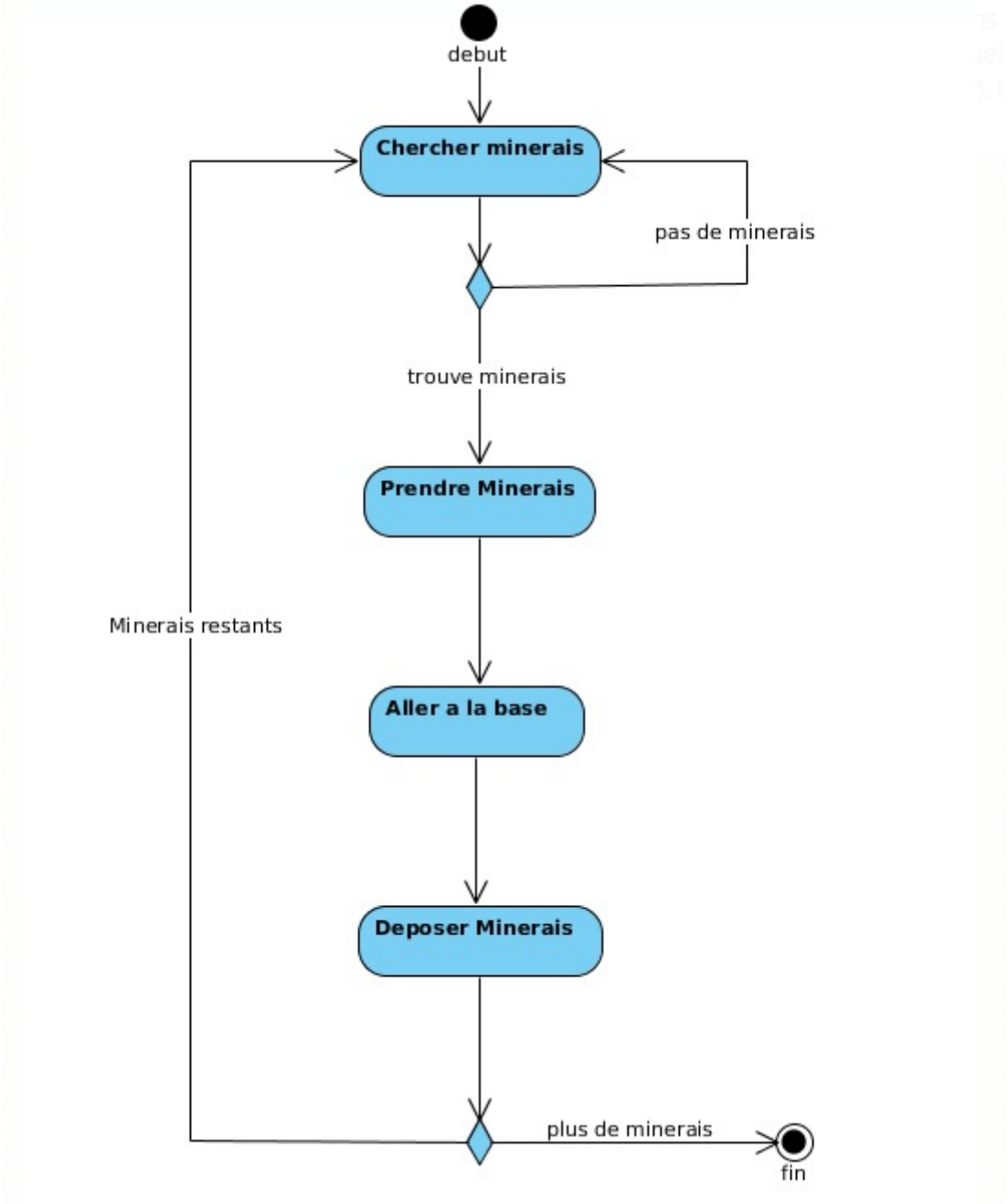


FIGURE 2.2 – Diagramme d'activités du robot réactif

Voici le diagramme d'activités du robot réactif. Ces activités correspondent à des réflexes de nos agents. Nous allons détailler chacune de ces activités.

2.2.1 Recherche de minerai [Chercher Minerais]

Dans cette étape du scénario le robot vérifie d'abord si la cellule de grille (qui correspond à une zone) dans laquelle il se trouve contient un minerais. Sinon il cherche si les cellules voisines en contiennent. Si ni la cellule actuelle ni aucune de ses voisines ne contiennent de minerais alors le robot se déplace aléatoirement dans l'une des cellules voisines de la cellule actuelle pour y exécuter le même traitement au prochain pas de temps (cycle). Dans le cas contraire (il trouve un minerais) il passe à l'étape de collecte du minerais.
Cette étape du scénario est implémentée de la façon suivante :

```
reflex chercher when: (objectif = "chercher") {
    write "Recherche de minerais";
    list<test_grid>  voisins <- position_actuelle.neighbours;
    bool trouv <- false;
    bool voisin_mine <- false;
    location <- position_actuelle.location;
    test_grid cellule <- position_actuelle;
    if (cellule.est_mine){
        cellule.color <- #blue;
        minerai_actuel <- cellule;
        trouv <- true;
        write "Minerai trouvé";
        objectif <- "prendre";
    }
    /* si la cellule actuelle ne contient pas de minerai on recherche parmi les voisins */
    if(! trouv){
        loop voisin over: voisins {
            if (voisin.est_mine){
                voisin_mine <- true;
                voisin.color <- #blue;
                minerai_actuel <- voisin;
                location <- voisin.location;
                write "Minerai trouvé";
                objectif <- "prendre";
                break;
            }
        }
    }
    /* s il n y a aucun voisin mine */
    if(!voisin_mine){
        position_actuelle <- one_of(voisins);
    }

}
```

FIGURE 2.3 – Réflexe chercher

2.2.2 Prise de minerai [Prendre Minerai]

Cette étape ne se déclenche que quand le robot trouve un minerais dans l'une des cellules qu'il vient d'explorer. Durant cette etape le robot correspond a la collecte de minerais presénts dans la zone explorée. Cela correspond dans notre simulation à la modification d'un certain nombre d'attributs du robot et de la cellule de grille dans laquelle le minerais a été trouvé.

Cette étape du scénario est implémentée de la façon suivante :

```
reflex prendre when: (objectif="prendre") {
    write "Extraction du minerai";
    minerai_actuel.color <- #red;
    minerai_actuel.est_mine <- false;
    position_actuelle <- minerai_actuel;
    objectif <- "aller_base";
}
```

FIGURE 2.4 – Réflexe prendre

2.2.3 Acheminement des minerais vers la base [Aller a la Base]

Cette étape correspond à l'ensemble des déplacements aléatoires que le robot fait de voisins en voisins afin de trouver la base et d'y déposer le minerais collecté.
Cette étape du scénario est implémentée de la façon suivante :

```
reflex aller_base when: (objectif="aller_base") {
    location <- position_actuelle.location;
    list<test_grid>  voisins <- position_actuelle.neighbours;
    bool voisin_est_base <-false;
    /* verifier si on est a la base */
    if(one_of(base_species).location= location){
        minerai_actuel.color <- #white;
        objectif <- "deposer";
    }
    else{
        /* verifier si la base est parmi les voisins */
        loop voisin over:voisins{
            if(one_of(base_species).location = voisin.location){
                position_actuelle <- voisin;
                voisin_est_base <- true;
                minerai_actuel.color <- #white;
                objectif <- "deposer";
                break;
            }
        }
    }
    /* si on n a pas trouver parmi les voisins une base; */
    if(!voisin_est_base){
        position_actuelle <- one_of(voisins);
    }

}
```

FIGURE 2.5 – Réflexe aller_base

2.2.4 Dépôt de minerai

Cette étape se déclenche une fois que le robot arrive a la base avec du minerais collecté. Elle correspond aussi (dans notre simulation) a la modification d'un ensemble d'attributs du robot qui montre que le dépôt a été bien fait et que le robot est à présent vide et prêt à aller rechercher d'autres minerais s'ils existent.

Cette étape du scénario est implémentée de la façon suivante :

```
reflex deposer when: (objectif="deposer") {
    write "Arrivée à la base";
    location <- position_actuelle.location;
    do depot;
}

action depot{
    nbMinesDeposes <- nbMinesDeposes + 1;
    write "Dépot d'un minerai effectué : nombre de minerais déposés = " + nbMinesDeposes;
    objectif <- "chercher";

}
```

FIGURE 2.6 – Réflexe deposer

Dans le diagramme d'activités ci-dessus les activités correspondent aux étapes que nous avons définies précédemment. Les seules nouveautés sont les deux nœuds de conditions :

- La première se fait après l'étape "Chercher Minerais" et permet de tester si le robot a trouver un minerais dans sa zone de perception et ainsi decider de s'il doit collecter et aller à la base ou tout simplement retourner à l'étape de recherche dans une autre zone de perception voisine.
- La seconde permet tester s'il existe encore des minerais dans l'environnement. Si c'est le cas le robot reprend tout le cycle d'activités à partir du début. Dans le cas contraire il s'arrête tout simplement.

Chapitre 3

Deuxième Version : Robot Cognitif

Les robots cognitifs sont tout d'abord des robots interactifs. La seule différence c'est qu'ils sont dotés de la capacité de se mémoriser la position de la base après l'avoir trouvé. Cette différence sera d'ailleurs la seule que l'on pourra remarquer dans les diagrammes des deux cotés.

3.1 Diagramme de Classes

Le diagramme de classe aussi ne contient pas beaucoup de changements pour les robots cognitifs. En plus du diagramme de classe précédent, nous avons une nouvelle classe "smart_species" (robot cognitif) qui hérite de la classe "test_species" (robot réactif) avec un seul nouvel attribut : "la_base". En effet il contient les coordonnées de la base permettant ainsi au robot de pouvoir s'y diriger directement. La méthode aller_base() y est redéfinie.

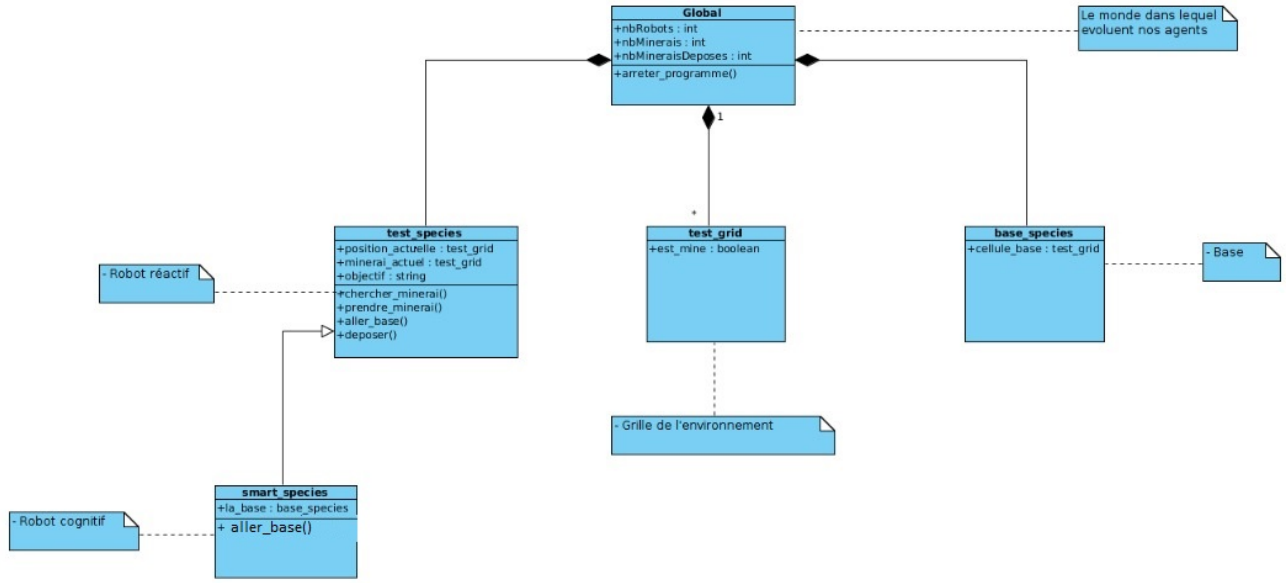


FIGURE 3.1 – Réflexe deposer

3.2 Diagramme d'Activités

Le diagramme d'activités du Robot Cognitif est le même que celui du robot réactif mis à part le fait que l'activité "Aller à la base" ne se fait pas de la même façon.

3.2.1 Acheminement des minerais vers la base [Aller a la Base]

En effet le déplacement aléatoire ne se fait que lors du premier voyage de transport du minerai vers la base. Après le dépôt le robot sauvegarde les coordonnées de la base pour s'y diriger directement dans le futur.

```
reflex aller_base when: (objectif="aller_base") {
    location <- position_actuelle.location;
    list<test_grid>  voisins <- position_actuelle.neighbours;
    bool voisin_est_base <-false;
    /* verifier si on est a la base */
    if(one_of(base_species).location = location){
        minerai_actuel.color <- #white;
        objectif <- "deposer";
    }
    else{
        /* verifier si la base est parmi les voisins */
        loop voisin over:voisins{
            if(one_of(base_species).location = voisin.location){
                position_actuelle <- voisin;
                voisin_est_base <- true;
                minerai_actuel.color <- #white;
                objectif <- "deposer";
                break;
            }
        }
    }
    /* si on n a pas trouver parmi les voisins une base; */
    if(!voisin_est_base){
        if(premiere_fois){
            position_actuelle <- one_of(voisins);
        }
        else{
            position_actuelle <- voisins closest_to la_base;
        }
    }
}
```

FIGURE 3.2 – Réflexe aller_base - Version cognitive

Chapitre 4

Tests de Sensibilité

Dans cette partie nous cherchons à étudier l'évolution du temps de récupération de tous les minerais de l'environnement en fonction du nombre de robots introduits.
Le nombre de minerais dans l'environnement est fixé à 40. Étant donné que certains paramètres rentrant en jeu dans l'évaluation de ce temps sont aléatoires (comme exemples, le temps pour aller à la base pour la version réactive, le temps pour trouver la base la première fois pour la version cognitive ou encore le temps pour trouver un minerai), nous répétons la même expérience avec le même nombre de robots un certain nombre de fois (dix ici) et prenons la moyenne des temps des expériences. Cela nous permet d'avoir des valeurs plus fiables. Ensuite, le programme enregistre ces valeurs dans un fichier. Ainsi pour chaque version, un fichier Excel est produit (*reactif.csv* et *cognitif.csv*). Ce sont ces tableaux Excel que nous exploitons par la suite.

```
experiment 'Run 10 simulations' type: batch repeat: 10 keep_seed: true {
    int nbSimulationPerNbRobots <- 10;
    int cpt <- 1;
    float moy <- 0.0;
    action _step_ {
        write "Step " + cpt;
        write "Step time : " + time;
        moy <- moy+time;
        if (cpt= nbSimulationPerNbRobots) {
            moy <- moy/nbSimulationPerNbRobots;
            save ["1", nbMinerais, nbRobots, moy, nbSimulationPerNbRobots] to:"reactif.csv" type: "csv
            write moy;
        }
        cpt <- cpt + 1;
    }
}
```

FIGURE 4.1 – Simulations répétées de la version réactive

Version	Nombre de minerais	Nombre de robots	Nombre de simulations	Temps moyen
1	40	1	10	87838.6
1	40	5	10	19608.0
1	40	10	10	10757.3
1	40	15	10	10835.4
1	40	20	10	10004.5

FIGURE 4.2 – Résultat d'exécution des simulations répétées de la version réactive

```
experiment 'Run 10 simulations' type: batch repeat: 10 keep_seed: true {
    int nbSimulationPerNbRobots <- 10;
    int cpt <- 1;
    float moy <- 0.0;

    action _step_ {
        write "Step " + cpt;
        write "Step time : " + time;

        moy <- moy+time;
        if (cpt= nbSimulationPerNbRobots) {
            moy <- moy/nbSimulationPerNbRobots;
            write moy;
            save ["2", nbMinerais, nbRobots, moy, nbSimulationPerNbRobots] to: "cognitif.csv" type: "c
        }
        cpt <- cpt + 1;
    }
}
```

FIGURE 4.3 – Simulations répétées de la version cognitive

Version	Nombre de minerais	Nombre de robots	Nombre de simulations	Temps moyen
2	40	1	10	14622.4
2	40	2	10	7489.9
2	40	3	10	7415.1
2	40	4	10	6346.8
2	40	5	10	5824.3
2	40	6	10	4466.4
2	40	7	10	5082.3
2	40	8	10	4696.7
2	40	10	10	5153.0
2	40	12	10	4610.8
2	40	15	10	6038.4

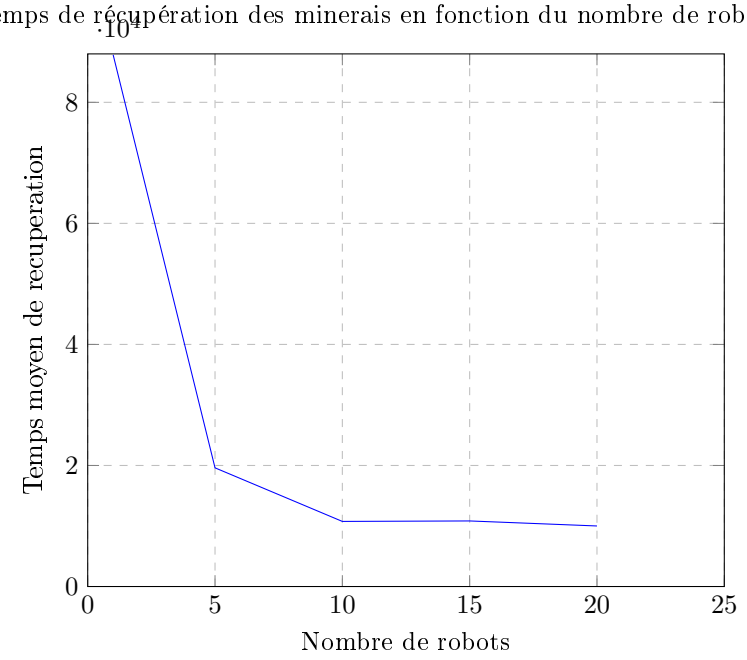
FIGURE 4.4 – Résultat d'exécution des simulations répétées de la version cognitive

4.1 Robots Réactifs

Version	Nombre de minerais	Nombre de robots	Nombre de simulations	Temps moyen
1	40	01	10	87838,6
1	40	05	10	19608,0
1	40	10	10	10757,3
1	40	15	10	10835,4
1	40	20	10	10004,5

TABLE 4.1 – Tableau récapitulatif du fichier reactif.csv

Evolution du temps de récupération des minerais en fonction du nombre de robots - Version réactive



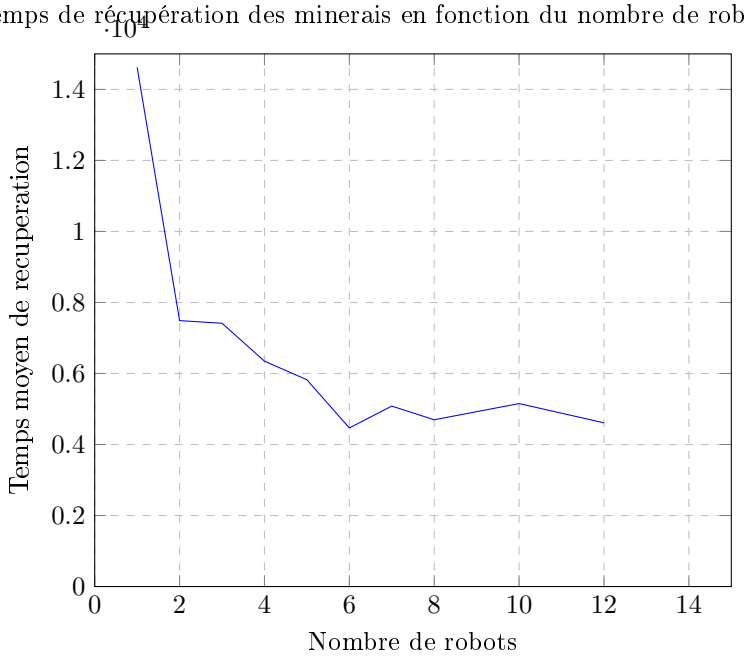
Interprétation
Nous remarquons que le temps moyen de récupération de tous les minerais diminuent au fur et à mesure que l'on ajoute plus de robots. Cependant, à partir de 10 robots, ce temps devient constant.
Concrètement, cela veut dire qu'à partir de 10 robots, il ne sert plus à rien d'augmenter le nombre de robots. Cela ne fait que gaspiller des ressources.
Aussi, nous notons que le temps optimal pour les robots réactifs tourne autour de 10000 cycles.

4.2 Robots Cognitifs

Version	Nombre de minerais	Nombre de robots	Nombre de simulations	Temps moyen
2	40	01	10	14622,4
2	40	02	10	7489,9
2	40	03	10	7415,1
2	40	04	10	6346,8
2	40	05	10	5824,3
2	40	06	10	4466,4
2	40	07	10	5082,3
2	40	08	10	4696,7
2	40	10	10	5153,0
2	40	12	10	4610,8
2	40	15	10	6038,4

TABLE 4.2 – Tableau récapitulatif du fichier cognitif.csv

Evolution du temps de récupération des minerais en fonction du nombre de robots - Version cognitive



Interprétation
Ici aussi, nous remarquons que le temps moyen de récupération de tous les minerais diminuent au fur et à mesure que l'on ajoute plus de robots. Cependant, ici dès qu'on atteint 6 robots, ce temps devient presque constant. A partir de 6 robots, il n'est plus nécessaire d'ajouter des robots
Aussi, nous notons que le temps optimal pour les robots cognitifs tourne autour de 5000 cycles.

4.3 Version réactive *vs* Version cognitive

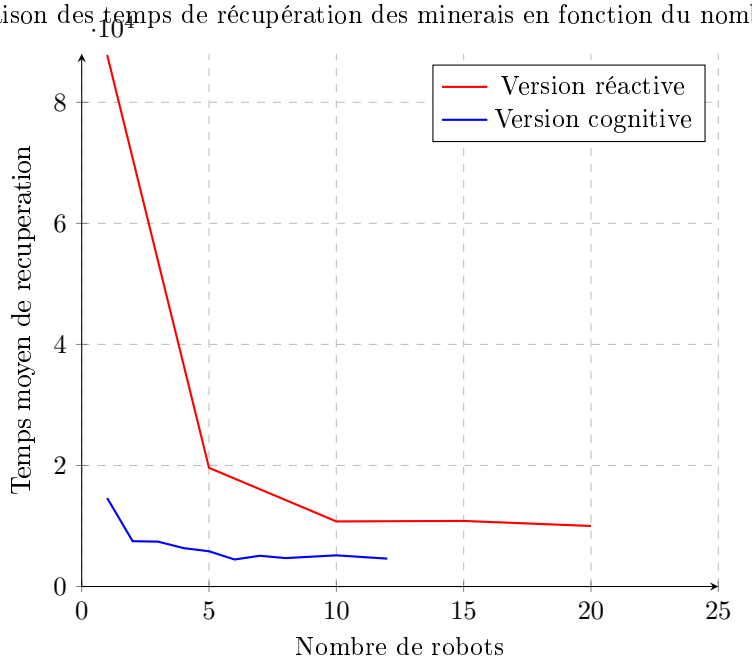
Version	Nombre de minerais	Nombre de robots	Temps moyen
Réactive	40	01	87838,6
Cognitive	40	01	14622,4

TABLE 4.3 – Temps de résolution pour un robot de chaque version

$$\frac{87838,6}{14622,4} = 6,0071$$

Nous remarquons qu'un robot cognitif met approximativement 6 fois moins de temps à récupérer tous les minerais qu'un robot réactif.

Comparaison des temps de récupération des minerais en fonction du nombre de robots



De même, nous remarquons que pour atteindre le temps optimal pour la version réactive, il nous faut introduire jusqu'à 10 robots alors que pour la version cognitive ce temps est atteint à partir de 6 robots. Aussi, à temps optimaux et nombres de robots optimaux, la version cognitive est 2 fois plus rapides (5000 cycles contre 10000 pour la version réactive).
Néanmoins, il faut noter que la version cognitive est plus gourmande en ressources alors que la version réactive consomme peu de ressources.
Ainsi, nous voyons que chaque version est adapté à une situation bien déterminée. En effet, si on a la possibilité d'avoir plusieurs et que l'on est limité en ressources, il est préférable d'utiliser la version réactive alors que si on est dans une situation ou les ressources ne constituent pas une contrainte, il est plus judicieux d'utiliser la version cognitive qui nous permet d'être plus efficace.

Conclusion

Dans cette étude de cas, nous étions face à la situation : l'exploitation d'un territoire minier par une population de robots. Pour y apporter des réponse, nous avons produits deux version de robot. Une dite réactive et une autre un peu plus "intelligente" dite cognitive. Nous avons vu que chacune de ces versions était adapté à une situation précise avec ses forces et faiblesses. Et si on faisait travailler ensemble ces deux types de robot ? Voilà une question qui mériterait d'être étudiée.

List des figures

Liste des Tables