Projet de FOSYMA
Wumpus Multi-agent

B.Thanh Luong, Gualtiero Mottola

Table des matières

1. Introduction

Ce projet Consiste a développer une version multi-agent d'un jeu Fortement inspiré de "Hunt the Wumpus" cette variante du jeu est définie de la façon suivante : un ensemble d'agents en coopération sont placé dans un environnement inconnu on pour mission d'explorer cet environnement et de récupérer un maximum de trésors qui sont disséminé dans cet environnement. Un agent Wumpus se trouve également dans l'environnement, il se déplace aléatoirement et a pour but de gêner l'exploration et la récupération des trésors.

2. Présentation des Agents

Les trois type d'agent utilisables pour récolter un maximum de Trésors sur la carte sont les suivants : les Agents Explorateurs qui n'ont pas la possibilité de récupérer des ressources, leur seul but est d'explorer la carte, des Agents Collecteurs qui ont un sac à dos correspondant a un type de trésor (`TREASURE` ou `DIAMONDS`) et qui ont une méthode permettant de récupérer ce type de trésor et le placer dans leur sac si celui-ci n'est pas plein. On note que lorsque cette action est exécutée une partie du trésor est perdue. Enfin le dernier type d'agent, l'agent Tanker qui ne peut pas ramasser de trésor mais a un sac a dos de capacité illimité, tous les agents collecteurs ont la possibilité de donner leur trésors à l'agent tanker. Ce sont les quantités présentes dans l'agent Tanker qui seront comptabilisées à la fin de l'exécution.

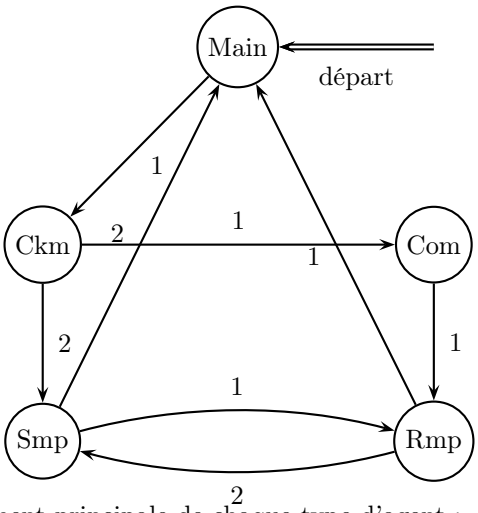
2.1. Comportement des Agents

Les comportements de nos trois types d'agents sont tous implémentés sous la forme de `FSMBehaviours` qui sont des automates finis. La classe offre des méthodes pour enregistrer les états et les transitions qui définissent l'ordre des behaviours. Chaque état du FSM est un behaviour qui est exécuté selon l'ordre définit par l'utilisateur.
Nous allons décrire dans cette section le comportement principal de chaque agent, puis dans la section suivante la suite de behaviours qui leur permet de communiquer et qui est identique pour tout les types d'agents. Le behaviour Main est le comportement principal de chaque agent, nous avons `ExploreBehaviour` pour les Agents Explorateurs, `CollectBehaviour` pour les agents Collecteurs, etc.. En général dans ce comportement, l'agent se déplace, collecte le trésor dans la case où il se trouve ou même ne rien fait mais surtout pas la communication avec les autres.

```
1  FSMBehaviour fsmBehaviour = new FSMBehaviour();
2  fsmBehaviour.registerFirstState(new MainBehavior(this),"Main");
3  fsmBehaviour.registerState(new CheckMailBehavior(this),"Ckm");
4  fsmBehaviour.registerState(new RequestConnectionBehaviour(this),"Com");
5  fsmBehaviour.registerState(new SendMapBehaviour(this),"Smp");
6  fsmBehaviour.registerState(new ReceiveMapBehaviour(this),"Rmp");
7
8  fsmBehaviour.registerTransition("Main","Ckm",1); //explore to check mail
9
10 fsmBehaviour.registerTransition("Ckm","Com",1); //check mail to start com
11 fsmBehaviour.registerTransition("Ckm","Smp",2); //check mail to send map
12
13 fsmBehaviour.registerTransition("Com","Rmp",1); //com to receive
14
15 fsmBehaviour.registerTransition("Smp","Rmp",1); // send to receive
16 fsmBehaviour.registerTransition("Smp","Main",2); // send to main
17
18 fsmBehaviour.registerTransition("Rmp","Main",1); // receive to explore
19 fsmBehaviour.registerTransition("Rmp","Smp",2); // receive to send
```

```
20
21 addBehaviour(fsmBehaviour);
```

Le schéma ci-dessous représente le comportement composé de nos agents :



Nous allons expliquer le comportement principale de chaque type d'agent :

Agent Explorateur : Sa mission principale est d'explorer la carte et d'établir une connaissance commune pour tous les agents. Il construit la carte au fur et à mesure dans sa propre table de hachage. Il met à jour sa structure de données personnalisée la disponibilité et la quantité des trésors. À la phase de la communication, tous les agents s'échangent leur carte pour compléter la connaissance. Une fois il a la carte complète, il va tenter d'aller vers les cases aléatoires dans la carte pour mettre à jour les trésors.

Agent Collecteur : L'agent ne fait que bouger aléatoirement dans la carte et ramasser les trésors de son type quand il n'a pas de carte complète. Si sa carte est complète, il va chercher des trésors dans l'ordre les plus proches de sa position actuelle puis les déposer au tanker.

Agent Tanker : Au début, il explore la carte comme les Agents Explorateurs pour accélérer le processus d'exploration. Quand il a la carte complète, il fixera sa position grâce à la centralisation dans le graphe pour que les Collecteurs puissent déposer le trésor. Le concept de centralisation sera expliqué dans la section Outils (3.2).

3. Processus de Communication

3.1. Les Processus

Après chaque action dans l'environnement (`Main`), l'agent passe au `CheckMailBehaviour` pour regarder sa boîte aux lettres, s'il reçoit des demandes de communication, il enchaîne les communications et il fusionne sa carte avec celles des agents qui sont de communiquer avec lui. S'il n'a rien dans sa boîte après un certain temps (ce qui est 50ms dans ce cas), il envoie une demande de communication à tout le monde (`RequestConnectionBehavior`). Les agents peuvent se communiquer si et seulement s'ils sont dans le rayon de communication (c'est-à-dire une distance de 3).

Revenons au `CheckMailBehaviour`, l'agent a reçu des demandes, il passe au `SendMapBehavior` pour envoyer sa carte, puis `ReceiveMapBehaviour` pour mettre à jour sa carte, et enfin il revient

au `Main`.

S'il diffuse sa demande (`RequestConnectionBehaviour`), il passe au `ReceiveMapBehaviour` pour attendre des cartes des autres. Après un certain temps (50ms aussi dans ce comportement), s'il n'en reçoit aucune il revient au `Main`. Sinon il fusionne sa carte, envoie sa carte mise à jour aux autres, et revient au `Main`.

Les deux cas dans `CheckMailBehaviour` distinguent bien les comportements suivants de l'agent. C'est pour cela dans le schéma ci-dessus, il y a au moins une flèche sortante de chaque état qui représente les conditions vérifiant les états venant.

Dans le cadre du projet, seulement les objets sérialisables et les chaînes de caractères sont autorisés dans la communication. C'est la raison pour laquelle nous utilisons la table de hachage et une structure de données manipulant les trésors qui implémente les objets sérialisables.

3.2. Les Outils

Calcul de chemin : Nous utilisions l'algorithme de Dijkstra pour la recherche du plus court chemin vers les cases non explorée et les trésors.

Centralisation dans le graphe : Nous plaçons le Tanker dans la centralité intermédiaire du graphe (voir plus ici). Dans la plupart des cases cette méthode est très efficace. L'inconvénient est si le graphe se décompose en 2 grands sous-graphes de même taille qui se connectent par un noeud unique, une fois le Tanker est placé, les autres agents ne peuvent plus accéder à l'autre partie du graphe.

Une autre méthode a été proposée. Nous prenons le noeud ayant les plus descendants comme le centre. Nous allons éviter le blocage. Cependant ce calcul n'est pas uniforme. Si nous voulons synchroniser la position du Tanker, il faudra passer en communication qui double le nombre de messages communiqués.

3.3. Trucs pas implémenté

INTERBLOCAGE !!!

3.4. Amélioration

Nous pourrions réduire le nombre de messages envoyés en stockant les agents communiqués après avoir la complétion de la carte.

Si les agents sont dans un réseau des environnements, il faudra reconstruire la structure de donnée pour sauvegarder les cartes différentes ainsi que la situation de l'agent dans chaque carte.

4. Conclusion