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. lors de la phase de la communication, tous les agents s'échangent leur carte pour compléter la connaissance, On note que tous les nœuds de la carte on un timestamp cela nous permet lors de l'échange de la carte de sélectionner les nœuds les plus récents si ils sont présents dans les deux cartes. Une fois la carte complété, l'agent sélectionner des nœuds aléatoires dans la carte pour mettre à jour les trésors.

Agent Explorateur des nœuds plus vieux : Cet Agent a exactement le même comportement que l'agent explorateur avant la complétion de la carte, cependant l'orque celle-ci est complété son but est d'aller explorer les nœuds les plus vieux de la carte et non des nœuds sélectionnés au hasard.

Agent Collecteur : Avant la complétion de la carte cet agent a exactement le même comportement que les agents explorateurs. Si sa carte est complète, il va chercher les trésors de son type qui sont les plus proches de lui. si son sac a dos est plein ou bien qu'il ne trouve plus de trésors de son type sur la carte il vas alors se diriger vers le tanker pour y déposer son butin.

Agent Tanker : Avant la complétion de la carte cet agent l'agent explore la carte comme les Agents Explorateurs pour accélérer le processus d'exploration. Quand le tanker a la carte complète, il fixe sa position en calculant la Betweenness centrality de tous les nœuds du graphe pour que les Collecteurs puissent déposer le trésor. la Betweenness centrality sera expliqué plus en détails dans la section Outils (3.2).

3. Processus de Communication

3.1. Les Processus

Ici les comportement `Main` est definit comme le comportement principal de chaque agent, par exemple le `ExploreBehaviour` pour l'agent explorateur. 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 en communiquer avec lui. S'il n'a rien dans sa boîte après un certain temps, il envoie une demande de communication à tout le monde (`RequestConnectionBehavior`). On note que les agents peuvent communiquer si et seulement s'ils sont dans le rayon de communication (c'est-à-dire une distance de 2 nœuds du graphe).

Revenons au `CheckMailBehaviour`,lorsque l'agent a reçu des demandes, il passe au `SendMapBehavior` pour envoyer sa carte a l'autre agent avec qui il est en communication, après cette étape il attends de recevoir la carte de son partenaire dans `ReceiveMapBehaviour` si il reçoit la carte dans la limite de temps prédéterminé il pourras ensuite fusionner les deux cartes, et enfin revenir au `Main`.

Si l'agent n'a pas de demandes dans sa boite au lettres alors il diffuse sa demande grâce a (`RequestConnectionBehaviour`), il passe ensuite au `ReceiveMapBehaviour` pour attendre des cartes des autres. Après un certain temps, s'il n'en reçoit aucune il revient au `Main`. Sinon il enverra

sa carte a l'agent avec qui il est en communication dans le `SendMapBehavior` puis fusionneras les cartes et revient au `Main`.

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 une 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 en considérant des arêtes ayant toutes le même poids ,de ce fait le plus cours chemin sera celui qui a le moins d'arêtes. L'implémentation utilisée de cette Algorithme est celle fournie par graphStream. Quand au calcul du plus court chemin entre deux points. Quand a la recherche du plus court chemin vers plusieurs cases, on applique cet algorithmes vers chacune des cases et on prends le chemin avec la valeur la plus proche.

Pour empercher le blocage des agents explorateurs et collecteurs sur le tanker du au fait que celui ci ne bouge pas, Après que la position du tanker est déterminé nous enlevons ce nœud des graphes pour le calcul de Dijkstra ce qui permet aux agents de passer autour du Tanker.

Centralisation dans le graphe : Nous plaçons le Tanker sur le nœud de plus haute Betweenness centrality dans le graphe. c'est a dire le nœud qui est dans le plus grande nombre de plus cours chemins entre deux autres nœuds quelconques du graphe. pour faire ce calcul nous utilisons l'algorithme fournis par graphstream. Dans la plupart des cas cette méthode est très efficace. elle permet de minimiser le chemin que les agents collecteurs ont a faire entre leur trésor et le tanker. L'inconvénient cependant, est que si le graphe se décompose en 2 grands sous-graphes de même taille qui se connectent par un nœud unique, une fois le Tanker est placé il est possible qu'il se trouve sur ce nœud unique, les agents ne peuvent donc plus accéder à la partie du graphe dans laquelle ils ne sont pas partie du graphe.

Une autre méthode a été proposée. prendre le nœud ayant les plus grand nombre de descendants.ce qui permet d'éviter Nous allons éviter le blocage du cas précédent. Cependant cela nécessiterais de communiquer la position du tanker a tous les agents. ce qui nous semblais être une moins bonne stratégie.

3.3. Interblocage

nous voulions un système robuste pour la gestion de l'inter-blocage. Il nous semblais en effet très important de minimiser un maximum les risque que deux agents se veuillent se diriger vers une case occupé par un autre agent, c'est pour cette raison que nous avons conçus notre système de communication de façon a ce qu'un échange de carte de fasse le plus souvent possible. Bien ce cette méthode soit peu économe en messages elle minimise les interblocages dans la phase d'exploration car les agents ne se dirigent pas vers les nœuds qui se trouvent déjà dans leur carte. Nous exploitons cette fonctionnalité de la façon suivante : lorsque deux agents se trouvent a proximité de deux cases, ils échangent leurs carte respectives et de ce fait ne se dirigent pas vers la position ou se trouve l'autre agent en communication car celle-ci sera marque comme exploré.

La méthode décrite ci dessus ne permet cependant pas d'éviter les interblocages dans la phase de récupération des trésors, pour contrer cela lorsque l'un de nos agents n'arrive pas a faire un mouvement, il vas sélectionner une case au hasard dans ses voisin et essayer de s'y déplacer jusqu'à ce que ce soit possible.

3.4. Amélioration

Nous pensons qu'il serais possible de grandement améliorer la gestion des interblocages, en effets la sélection d'une case au hasard lorsqu'un agent se bloque est sous optimale lorsque les agents se croisent dans une ligne.

Pour accélérer la récupération des Trésors il serais possible d'augmenter le nombre de tanker cela pourrais potentiellement diminuer le temps de trajet des agents des trésors vers le Tanker, on pourrais aussi imaginer un tanker qui se déplace vers les gants collecteurs.

Nous avons aussi penser a réorienter les agents explorateurs après la complétion de la carte pour bloquer l'agent wumpus, cela nous permettrais de ne pas nous soucier de la mise a jour de la carte après l'exportation

Si les agents sont lâchés dans un réseau composé de différents 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

Vous trouverez ci dessous Les performances de notre algorithmes sur l'instance de l'examen de 2017

Instance2017	Trésors	Diamants
Total	280	140
Ramassé	202	128

E n conclusion fonctionne rmassage rapide exploration tres rapide vulnerabilités bloquage sous optimale