Projet de FOSYMA
Wumpus Multi-agent

B.Thanh Luong, Gualtiero Mottola

Table des matières

1 Introduction **2**

2 Présentation des Agents **2**
2.1 Comportement des Agents 2

3 Communication et interblocage **3**
3.1 Processus de Communication 3
3.2 Les Outils 6
3.3 Interblocage 7

4 Conclusion **7**

1. Introduction

Ce projet consiste à 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. Ils ont pour mission d'explorer cet environnement et de récupérer un maximum de trésors qui sont disséminés 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 à dos de capacité illimitée, tous les agents collecteurs ont la possibilité de donner leurs 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 fini. La classe offre des méthodes pour enregistrer les états et les transitions qui définissent l'ordre des behaviours. Chaque état de l'automate fini est un behaviour qui est exécuté selon l'ordre définit par l'utilisateur.

voici un exemple du code de l'automate fini de notre agent explorateur.

```
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); //main 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);
```

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 tous les types d'agents.

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 dans 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 sont horodaté, cela nous permet lors de l'échange de la carte de sélectionner les nœuds les plus récents s'ils sont présents dans les deux cartes. Une fois la carte est complète, l'agent sélectionne des nœuds aléatoires dans la carte pour mettre à jour leur contenu, tel que la quantité de trésors présente.

Agent Explorateur des nœuds plus vieux : Cet agent a exactement le même comportement que l'agent explorateur avant la complétion de la carte, cependant lorsque celle-ci est complète 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. Lorsque sa carte est complète, il se dirige vers les trésors de son type qui sont les plus proches de lui. Si son sac à 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 la carte ne possède plus de nœuds inexploré , Le Tanker vas alors fixer sa position en calculant la Betweenness centrality de tous les nœuds du graphe, puis sélectionner nœuds ayant la valeur la plus grande. Cette méthode permet aussi aux Collecteurs de calculer la position du Tanker pour qu'ils puissent y déposer leurs trésor. Le calcul de la Betweenness centrality sera expliqué plus en détails dans la section Outils (3.2).

3. Communication et interblocage

3.1. Processus de Communication

Notre algorithme de communication que nous décrivons ci-dessous a pour but maximiser le nombre de communications entre agents, nous ferons donc tout d'abord une description de l'algorithme, puis une analyse du nombre de messages échangés par un agent lors d'une discussion.

Ici le comportement `Main` est définit comme le comportement principal de chaque agent, par exemple le `ExploreBehaviour` pour l'agent explorateur. Description de l'algorithme :

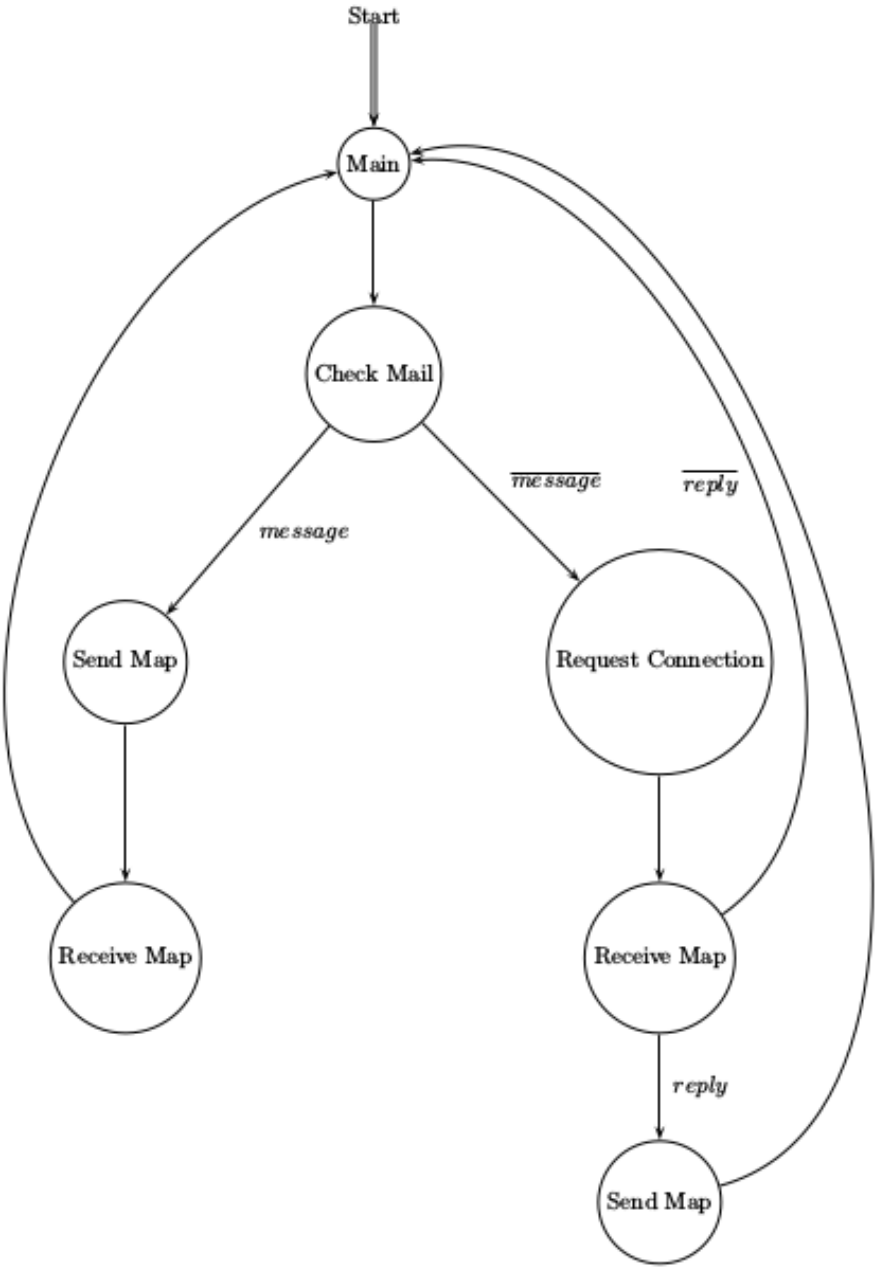
- Après chaque action dans l'environnement dans le Behavior (`Main`), l'agent passe au `CheckMailBehaviour` pour regarder sa boîte aux lettres. On distingue le cas ou il reçoit des demandes de communication et ou il ne les reçois pas.
- Si l'agent n'a rien dans sa boîte après un temps d'attente prédéfinit de 50ms , il envoie une demande de communication à tout les autres Agents avec (`RequestConnectionBehavior`). il passe ensuite au `ReceiveMapBehaviour` pour attendre les cartes des autres agents. Après 50 autres millisecondes, s'il n'en reçoit aucune il revient au `Main`. Sinon il enverra sa carte à l'agent avec qui il est en communication dans le `SendMapBehavior` puis fusionnera les cartes pour ensuite revenir au `Main`.

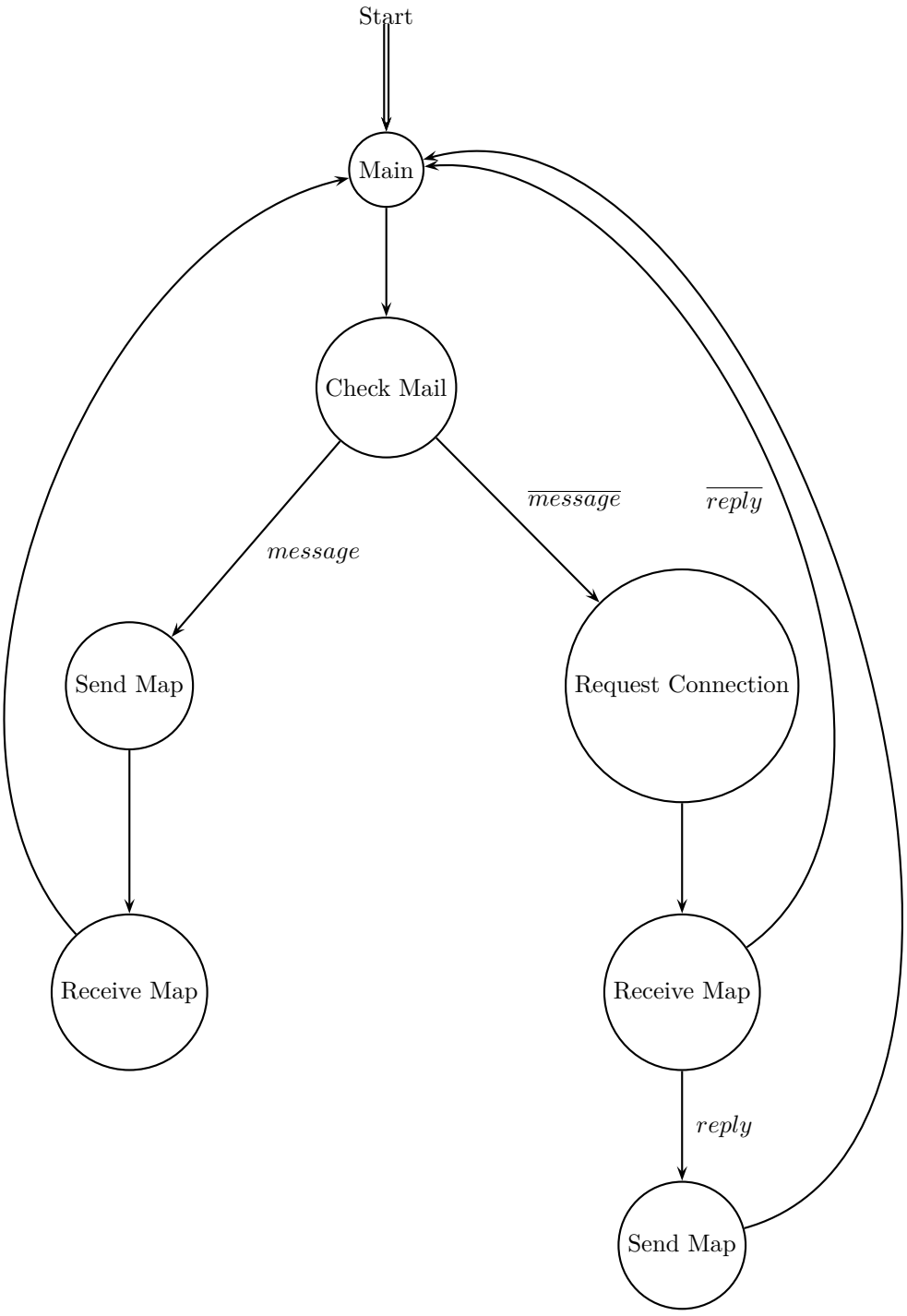
— Si l'agent a un message et que celui-ci est moins vieux que 50ms, il passe alors au `SendMapBehavior` pour envoyer sa carte à l'agent dont il récupère le nom dans le message reçu précédemment. Après cette étape il attend de recevoir la carte de son partenaire dans `ReceiveMapBehaviour` s'il reçoit la carte dans la limite de temps prédéterminé il pourra ensuite fusionner les deux cartes, et enfin revenir 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 pour représenter la carte . Cette Structure sérialisables contiens notamment des informations sur les trésors et la date de découverte des nœuds. Nous terminons donc par l'analyse du nombre de messages échangé par un agent dans le pire des cas, nous prenons donc le cas ou l'agent ne trouve pas de messages dans sa boite aux lettres :

— Envoie d'un message à tous les autres agents sur la carte (**n** Agents)
— Envoie de la carte à l'agent avec qui il est en communication

On note donc que le nombre de messages envoyés dans le pire des cas à chaque communication est **n+1**. De plus, après chaque mouvement, même si aucun agent ne se trouve dans le rayon de communication de l'agent en question, **n** messages serons envoyé. Il serait donc possible de grandement réduire le nombre de messages en débutant la communication lors du blocage de l'agent, mais du fait que les agents peuvent communiquer s'ils sont dans le rayon de communication (c'est-à-dire une distance de 2 nœuds du graphe), cela réduirait grandement le nombre d'échange de cartes, ce qui ne nous semblait pas convenir à notre stratégie d'exploration.


Start
Main
Check Mail
$\overline{message}$
$\overline{reply}$
$message$
Send Map
Request Connection
Receive Map
Receive Map
$reply$
Send Map



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`. Quant à la recherche du

plus court chemin vers plusieurs cases, on applique cet algorithmes vers chacune des cases et on sélectionne le chemin avec la valeur la plus proche.

On note qu'après le calcul de la position du tanker, nous enlevons le nœud des graphes pour le calcul de Dijkstra, cela permet d'empêcher le blocage des agents explorateurs et collecteurs sur le tanker du fait que celui-ci ne se déplace pas. Les autres agents passent donc autour du Tanker.

La Complexité de l'algorithme de Dijkstra est enO(n log(n) + m) avec n le nombre de nœuds et m nombre d'arêtes.

Centralisation dans le graphe : Nous plaçons le Tanker sur le nœud de plus haute Betweenness centrality dans le graphe, c'est-à-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 fourni 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 que 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 de laquelle ils ne font pas partie.

Une autre méthode a été proposée : de prendre le nœud ayant les plus grand nombre de descendants, ce qui permet d'éviter le blocage du cas précédent. Cependant cela nécessiterais de communiquer la position du tanker a tous les agents, ce qui nous semblais être plus coûteux en terme de communication et une moins bonne stratégie. **L**a Complexité de l'algorithme du calculd de la Betweenness centrality est enO(nm).

3.3. Interblocage

Nous voulions un système robuste pour la gestion de l'interblocage. Il nous semblait en effet très important de minimiser un maximum des risques que deux agents se veuillent se diriger vers une case occupée par un autre agent, c'est pour cette raison que nous avons conçu notre système de communication de façon à ce qu'un échange de carte de fasse le plus souvent possible. Bien que 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 à proximité de deux cases, ils échangent leurs carte respectives et de ce fait ne se dirigent pas vers la position où se trouve l'autre agent en communication car celle-ci sera marquée comme explorée.

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 à faire un mouvement, il va sélectionner une case au hasard parmi ses voisins et essayer de s'y déplacer jusqu'à ce que ce soit possible.

DEPENDANY DE LEXPLORATON TOTAL

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

En conclusion notre méthode remplie le critère de récupération des trésors sur la carte bien que nous remarquons un perte substantielle du nombre de trésors ramassé due a méthode de ramassage

et au déplacement des trésors par le wumpus. Les communications entre agents, cher en nombre de messages permettent un partage de l'information rapide entre nos agents. la position de notre tanker nous semble optimale, bien que le déplacement de celui ci nous permettrais sans doute de récupérer les trésors plus rapidement. de plus le tanker statique risque de créer des problèmes sur certaines cartes partagés en deux par un couloir. Notre approche vis a vis des interblocages pourrais se résumer de la façon suivante, nous essayons d'en minimiser le nombre avant que ceux ci arrivent, mais la gestion aléatoire des blocages est sous optimale.

Améliorations Nous pensons qu'il serait possible de grandement améliorer la gestion des interblocages, en effet 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 serait possible d'augmenter le nombre de Tanker cela pourrait potentiellement diminuer le temps de trajet des agents des trésors vers le Tanker, on pourrais aussi imaginer un Tanker qui se déplace vers les agents collecteurs.

Nous avons aussi pensé à réorienter les agents explorateurs après la complétion de la carte pour bloquer l'agent Wumpus, cela nous permettrait de ne pas nous soucier de la mise à jour de la carte après l'exportation