Sorbonne université

Rapport de projet FOSYMA : Dédale ("Hunt the Wumpus")

Étudiant :

Baptiste Jarry

Lien du dépôt : https://github.com/Batary/ProjetFOSYMA

Année 2017-2018

Table des matières

1 Introduction

"Hunt the Wumpus" de **Gregory Yob** est un des premiers jeux informatiques, sorti en 1972. Le but du jeu est (à l'origine) de deviner où se trouve le *Wumpus* (ou *golem*) sans jamais le rencontrer. Dans la version multi-agents modifiée du projet, le but est de ramasser le plus de trésor possible, le plus rapidement possible. Le golem n'est pas dangereux mais il peut déplacer les trésors (en perdant une partie au passage) et bloquer nos agents.

L'essentiel du temps alloué au projet a été consacré à la gestion des inter-blocages et aux algorithmes de recherche de chemin avancés. Les autres aspects du jeu se basent sur ces algorithmes.

2 Stratégies des agents

2.1 Exploration

Les agents collecteurs et explorateurs se chargent de l'exploration. L'agent silo, quant à lui sert de point de repère aux autres agents.

Un agent qui explore tente d'atteindre le nœud non visité le plus proche, en pénalisant les nœuds ayant d'autres agents en cours d'exploration plus proches d'eux.
Avantage : l'exploration est très rapide dans les parties du graphe qui sont des couloirs ou des zones avec peu de nœuds.
Inconvénient : l'exploration est parfois moins efficace dans le cas de zones très connectées avec beaucoup de nœuds.

Lorsqu'un trésor est trouvé, l'agent revient vers l'agent silo pour partager sa trouvaille et également prendre connaissance des mises à jour des autres agents. Cela limite le cas où un agent explore de nouveau une zone déjà explorée par manque d'échanges entre les agents.

Lorsque l'exploration est terminée, les agents parcourent la carte en cherchant à mettre à jour les endroits visités depuis le plus longtemps afin de trouver d'éventuels trésors déplacés.

Complexité

Pour un agent donné, à chaque étape **k**, l'agent parcourt au maximum **k-1** nœuds. On a donc un majorant approximatif du nombre total de nœuds parcourus pour un graphe de **n** nœuds égal à $1 + 2 + 3 + ... + (n-1) = \frac{n(n-1)}{2}$

2.2 Communication

Toutes les **k** itérations, chaque agent envoie une mise à jour aux autres agents. Cette mise à jour est un objet conteneur sérialisé qui contient les informations utiles connues par l'agent : la carte, l'état des trésors et les agents. Il s'agit donc d'un objet "volumineux", mais envoyé avec une fréquence faible.

Ces messages constituent la quasi-totalité des messages envoyés. En effet, l'idée est qu'avec suffisamment d'informations, il n'y a pas besoin d'autres échanges car les agents peuvent prendre en compte ce que font les autres. Le contrecoup de cette manière de faire est que chaque agent doit effectuer davantage de calculs.

Complexité

Soit n_a le nombre d'agents et **t** le nombre de pas de temps total d'exécution du programme. Chaque agent envoie tous les **k** pas de temps un message aux $n_a - 1$ autres agents. Au total, on a donc $\frac{t}{k} n_a(n_a - 1)$ messages échangés au maximum.

2.3 Gestion des inter-blocages

Comme indiqué dans le paragraphe précédent, les agents sont au courant du chemin des autres. Cela permet de choisir son propre itinéraire en fonction du chemin des autres et de les laisser passer le cas échéant. Une priorité statique permet d'éviter que deux agents qui se rencontrent ne reculent en même temps pour laisser passer l'autre.

La fréquence d'envoi des messages augmente quand un agent est bloqué, afin que tous les agents aient un maximum d'informations et donc une estimation précise de la situation pour que le blocage dure le moins longtemps possible.

En cas de blocage persistant, un comportement spécifique dédié à la gestion des inter-blocages (*UnstuckBehaviour*) est activé et remplace le comportement actuel. Ce comportement effectue des mouvements aléatoires de l'agent et tente périodiquement (la période augmente à chaque fois) d'atteindre : 1) un nœud *non bloquant* (c'est à dire qu'en l'excluant il existe un chemin court entre chacun de ses voisins) ; ou bien : 2) son objectif initial.
Le comportement ne s'arrête que quand l'objectif initial a été atteint (d'une manière ou d'une autre).

2.4 Ramassage des trésors

Comme indiqué dans la partie *exploration*, un trésor trouvé est immédiatement reporté à l'agent silo, et une partie est déjà transférée si l'agent est un *collecteur*. Si le trésor est suffisamment important, un agent collecteur peut alors proposer à l'agent silo d'aller le chercher avec lui pour minimiser le temps de collecte. L'agent silo perdra alors temporairement son rôle de référence (les autres l'attendront au même endroit) au profit d'un ramassage très rapide.
Avantage : le ramassage des trésors est très efficace s'il y a peu d'agents collecteurs et si les trésors sont de grande valeur.
Inconvénient : le partage d'informations et la récupération passive des trésors sont momentanément suspendus. L'opération peut être mauvaise s'il y a beaucoup d'agents qui viennent déposer leur trésor ou au contraire peu d'agents venant par intermittence échanger des informations.

Lorsque le ramassage des trésors est terminé et que toute la carte est explorée, l'agent silo envoie un message à l'AMS pour stopper la simulation. Si cette fonctionnalité est désactivée, les agents continuent de parcourir la carte en quête d'un trésor qui aurait été déplacé.

2.5 Coordination

L'agent silo va essayer au début de la simulation de trouver un nœud *non bloquant* qui va permettre aux autres agents d'aller et venir sans être gênés.

Lorsqu'un message proposant de ramasser un trésor est reçu, l'agent *assigne* ce trésor à l'envoyeur et se met en route en même temps que lui pour le ramasser. Les autres agents collecteurs sont mis au courant de cette affectation et n'essaieront pas d'aller le ramasser pour éviter les blocages et le déplacement inutile de l'agent silo.

L'agent collecteur se place directement sur le trésor et attend que l'agent silo arrive. Celui-ci se met sur une case adjacente au trésor et attend de même l'agent collecteur, en prenant soin de ne pas lui bloquer le passage si le trésor est difficile d'accès.

Complexité

Un seul message, venant d'un collecteur pour proposer de ramasser un trésor, est envoyé en plus des messages habituels qui suffisent à gérer les autres cas.

2.6 Gestion du golem

La gestion du golem n'est pas optimale en raison du manque de temps pour terminer le programme. En effet, les agents ne pouvant pas savoir où il se trouve précisément, ils sont souvent bloqués. Ils ne peuvent le trouver que lorsqu'ils sont effectivement bloqués et prennent alors en compte sa position.
Les agents se retrouvent donc à aller et venir autour du golem. Celui-ci se déplaçant de façon aléatoire, il a tendance, lorsqu'il est coincé dans un couloir, à revenir en arrière même lorsqu'il a la possibilité de sortir.
Au final, les cas difficiles comme celui de la figure ci-dessous mettent parfois un temps assez important à se résoudre, même si les agents laissent régulièrement au golem la possibilité de sortir.

FIGURE 1 – Un exemple de blocage.

3 Conclusion

Bien que le temps ait manqué pour pouvoir finir le polissage du programme, les agents fonctionnent de manière efficace et parviennent à ramasser la quasi-totalité des trésors sans problème.

3.1 Avantages de l'implémentation

Les agents échangent peu de messages, et évitent les inter-blocages en échangeant suffisamment d'informations. Toutes les tâches peuvent être effectuées sans avoir besoin d'une coordination explicite entre les agents. Les environnements contenant des trésors de grande valeur avec peu d'agents sont traités très rapidement.

3.2 Inconvénients

Les messages échangés sont volumineux, et le calcul nécessaire pour trouver un chemin est assez important. L'agent silo étant statique, un environnement avec beaucoup de petits trésors ne sera pas traité de manière optimale, car tous les collecteurs devront faire un aller-retour à chaque ramassage vers un point qui peut être éloigné (voir section *Améliorations possibles*).

3.3 Partis pris

En raison de l'inconsistance des informations qu'un agent possède, beaucoup de suppositions ont du être faites, notamment :

— Quand on laisse passer un agent ou qu'on veut tenir compte de sa trajectoire, on suppose qu'il ne sera pas bloqué.

— Beaucoup de valeurs ont été définies directement dans le code : temps d'expiration du statut d'un agent, pas de temps initial pour la gestion des inter-blocages (voir section dédiée), etc.

— Lorsqu'un trésor est ramassé par le golem, il est difficile de savoir pour les agents s'il a été ramassé par un autre agent ou par le golem. Quand on vérifie que la simulation est terminée (carte explorée + sacs vides + tous les trésors ramassés), on ne peut donc pas savoir si le golem a bougé un trésor ; cette possibilité n'est donc pas prise en compte.

Dans le code, un comportement (*MoveBehaviour*) permet de gérer spécifiquement le déplacement. Il est piloté par les comportements "décideurs" qui lui indiquent le chemin à suivre. Ces comportements ne sont réveillés que quand un événement se produit : message reçu, trésor trouvé, destination atteinte, agent bloqué. Cela permet d'éviter les calculs superflus quand il ne se passe rien de nouveau.

3.4 Améliorations possibles

La gestion du golem n'est pas optimale : lorsqu'il est coincé, les agents pourraient essayer de lui laisser un passage vers un nœud *non bloquant*. Ils lui laissent actuellement un passage, mais de façon sporadique et le passage ne reste pas ouvert longtemps.
Une autre amélioration pourrait être de coordonner les agents pour que par exemple les explorateurs protègent les trésors en restant dessus ou bien en bloquant le golem (mais ici il y a un risque qu'il ait emporté un trésor avec lui).

Les trésors pourraient également être organisés par priorité, par exemple en premier lieu tenter de récupérer les trésors proches du golem, puis les trésors les plus intéressants (pour éviter les pertes dues au golem) ou proches (pour les récupérer plus vite et en avoir moins à surveiller).

Les informations envoyées pourraient être filtrées pour n'envoyer que ce que le destinataire ne sait pas encore (en s'appuyant sur le dernier message que l'on a reçu de lui). Cela réduirait fortement la taille des messages envoyés.

Les mises à jour sont effectuées en se basant sur le temps de la machine. Cette implémentation est efficace sur une seule machine mais ne fonctionne plus si les agents sont sur des machines séparées. Une manière de faire pourrait être d'utiliser une horloge matricielle pour connaître les temps d'un événement par rapport à chaque agent.

L'agent silo rejoint le premier nœud *non bloquant* qu'il trouve afin de pouvoir donner une référence aux autres agents le plus tôt possible et ne bouge plus ensuite. Une amélioration pourrait être de déplacer cet agent pour le rapprocher des trésors, en faisant attention à bien mettre à jour les références des autres agents.

Un seul agent silo a été prévu dans le code (note : son nom doit forcément être "*AgentTanker1*"). Cependant, ce dernier est facilement généralisable en transformant la référence unique en un ensemble de références qui seraient autant de points de communication et de récupération des trésors.