Projet de FOSYMA

MU4IN202 - Printemps 2021

Projet Dédale

Auteurs :
Youssef KADHI
Vincent LIM
Groupe 9

Encadrants :
M. Cédric HERPSON
Mme. Aurélie BEYNIER
M. Nicolas MAUDET

2 mai 2021

Table des matières

1 Introduction

L'objet de ce projet est de développer, à travers la plateforme JADE, une variante multi-agent du jeu « Hunt the Wumpus », un des premiers jeux informatiques, développé par Gregory Yobe en 1972. Dans la version d'origine, le joueur a pour objectif de bloquer un golem, le Wumpus, caché dans un labyrinthe tout en ramassant des trésors. Dans la variante implémentée, une équipe d'agents coopératifs doit explorer un environnement inconnu pour ensuite pouvoir traquer et bloquer le Wumpus. Les trésors à récupérer sont omis dans cette version.

Pour cela nous disposions de différents environnements (grilles, arbres...) qui sont modélisés par un graphe non-orienté dans lequel les sommets représentent les positions que peuvent occuper nos agents, et les arêtes représentent les déplacements possibles entre les positions. Chaque agent occupe un noeud à tout moment et un même sommet ne peut accueillir plusieurs agents.

Les agents se retrouvent dans un environnement inconnu. En effet, ils sont aveugles au nombre d'agents déployés et de la topologie de la carte au moment de leur initialisation. Néanmoins, ils peuvent observer les noeuds qui leur sont adjacents et communiquer avec les autres agents qui entrent dans leur portée de communication prédéfinie. De ce fait l'exploration de la carte et le partage d'information occupent un rôle primordial à leur initialisation.

Le Wumpus quant à lui, est un agent capable de se déplacer et qui émet une odeur (stench), observable par les agents, autour de lui. L'objectif des agents est donc d'explorer l'environnement afin de pouvoir ensuite bloquer le Wumpus le plus rapidement possible.

2 Finite State Machine

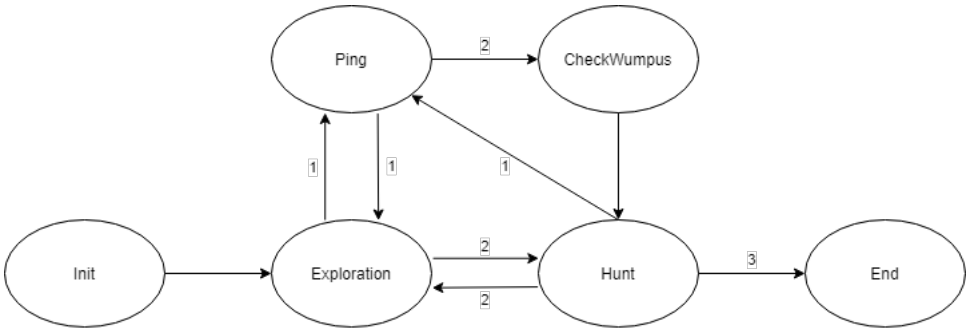

FIGURE 1 – FSM suivi par les agents

Dans cette partie, nous allons vous présenter le Finite State Machine (FSM) que suivent nos agents.

— **Init** est l'état initial, l'agent initialise sa carte.

— **Exploration** est l'état d'exploration dans lequel l'agent effectue un mouvement, il passe ensuite à l'état **Ping** par défaut, ou l'état **Hunt** s'il a détecté une odeur.

— **Hunt** est l'état de chasse dans lequel l'agent suit les stenchs. S'il en trouve, il se déplace vers la stench et passe à l'état **Ping**. S'il n'en trouve pas, il retourne à l'état **Exploration** si cette dernière n'est pas temriné. L'agent passe à l'état **End** s'il a bloqué un Wumpus.

— **Ping** est l'état qui permet à l'agent d'envoyer un ping dans ses alentours à la recherche d'agents, l'état redirige ensuite vers **Exploration** ou **CheckWumpus** en fonction de si l'agent est en mode exploration ou chasse.

— **CheckWumpus** est l'état qui permet à l'agent, en mode de chasse, de vérifier s'il a trouvé un Wumpus

— **End** est l'état finale qui indique que l'agent a bloqué un Wumpus.

3 Communication

Afin que les phases d'exploration et de chasse soient efficaces, il est primordial que les agents se coordonnent entre eux. Cette coordination se fait par le biais de communication. De ce fait, les agents s'enregistrent auprès d'un page jaune qui répertorie alors l'ensemble des agents présents dans l'environnement. Ils devront consulter cette liste afin de pouvoir identifier les destinataires de leurs pings. Les agents disposent d'un **ReceiveMessageBehaviour** qui permet de réceptionner les **4 types** de messages que les agents peuvent s'envoyer : un ping, une confirmation de réception du ping, une carte, une destination.

Les messages de type ping sont émis par les agents après chaque mouvement, l'intérêt est de pouvoir enregistrer les agents à proximité qui seront détectés lorsqu'ils répondent au ping.

Lorsqu'un agent reçoit un ping, il s'identifie en envoyant 3 informations : son id, son nom, et sa position.

Lorsqu'un agent reçoit une réponse à son ping, il enregistre le receveur dans sa liste d'agents détectée (qui est réinitialisée après chaque mouvement), et lui envoie sa carte. L'agent pourra alors passer par cette liste au lieu de la page jaune, afin de contacter les agents à proximité.

Lorsqu'un agent reçoit une carte, il la fusionne avec la sienne et met à jour sa liste de noeuds.

Lorsqu'un agent reçoit une destination (position contenant une odeur ou un Wumpus détecté), il se dirige vers cette position.

Après chaque mouvement, l'agent envoie $n-1$ (n le nombre d'agents dans l'environnement) pings afin de s'assurer de ne rater aucun agent. En effet, nous avions initialement considéré un envoie de ping à intervalle de temps régulier afin de limiter la quantité de pings envoyés. Néanmoins, les agents se déplaçant très rapidement, cela pourrait permettre à des agents assez proches de s'éloigner hors de leur portée de communication sans avoir pu communiquer et partager leur carte. Les pings étant envoyés après chaque mouvement, nous avons voulu limiter la taille des messages transmis. Ce sont, de ce fait, des messages vides.

Pour le partage de la carte, notre implémentation n'est pas optimale. En effet, chaque agent envoie sa carte à tous les autres agents à proximité. On a donc $k(k-1)$ échanges (k le nombre d'agents présent). Nous aurions voulu implémenter une méthode de centralisation des messages dans lequel les agents sélectionneraient un agent (celui qui a l'id le plus petit par exemple) qui recevrait toutes les cartes, les fusionneraient puis les renverraient aux agents. On aurait alors eu une complexité de $2k-4$ en nombre de messages.

4 Exploration

Les agents n'ayant initialement aucune connaissance sur l'environnement, le Behaviour d'exploration a naturellement été le premier à être implémenté. En effet, les agents ne peuvent pas traquer le Wumpus s'ils ne connaissent pas les positions possibles. Ce comportement est par ailleurs le comportement adopté par défaut lorsque les agents ne sont pas en train de traquer le Wumpus.

L'environnement étant modélisé par un graphe non orienté et sans poids, l'exploration est basée sur un algorithme de parcours en profondeur (DFS). De ce fait, l'agent dispose de 2 structures de données :

— Une liste **nodeList** de noeuds existants auquel un attribut **open** ou **closed** est associé en fonction de si le noeud a déjà été visité.

— Une liste d'arêtes afin de pouvoir relier les différents noeuds du graphe entre eux.

À chaque déplacement jusqu'à ne plus avoir de noeud ouvert :

1. L'agent change l'attribut du noeud où il se trouve en **closed**.
2. L'agent observe les cases adjacentes et les ajoutent à **nodeList** s'ils n'y sont pas.
3. L'agent calcule sa prochaine destination :
   — S'il y a des noeuds adjacents ouverts, il en choisit un et s'y déplace.
   — S'il n'y en a pas, on explore le graphe connu par un algorithme de parcours en largeur (BFS) jusqu'au premier noeud ouvert trouvé. On utilise ensuite l'algorithme de Dijkstra afin de calculer le chemin. Tant que ce noeud n'a pas été atteint, on récupère le noeud suivant à partir du chemin qui a été gardé en mémoire.

Lorsque plusieurs agents sont à proximité et s'échangent leur carte, il se peut qu'ils aient tous le même noeud ouvert le plus proche. Ils auront donc la même destination et se déplaceraient en file indienne. Afin d'éviter cela, les agents se coordonnent en fonction de leur id : ils se classent par ordre croissant de leur id et en fonction de leur position x dans la liste, ils se dirigeront vers le x-ième noeud ouvert le plus proche.

DFS et BFS ont une complexité en temps et en mémoire de $O(n+m)$ tandis que Dijktstra a une complexité en temps de $O(nlog(n)+m)$ et $O(n+m)$ en mémoire (n le nombre de noeud et m le nombre d'arêtes). Étant dans un graphe sans poids, utiliser seulement BFS comme algorithme de plus court chemin aurait été optimal, mais nous n'avons pas su enregistrer les successeurs et le chemin.

Après chaque déplacement, l'agent consulte les noeuds qui lui sont adjacents : s'il y trouve une odeur, il arrête son comportement d'exploration et passe au comportement de chasse.

5 Chasse

En mode de chasse, si l'agent ne détecte pas d'odeur dans les noeuds adjacents, il repasse en mode d'exploration s'il n'a pas déjà exploré l'intégralité de l'environnement. Si c'est le cas, il reste en mode chasse et se déplace aléatoirement jusqu'à détecter une odeur. S'il détecte des odeurs, il se déplace aléatoirement dans un des noeuds concernés.

Après chaque tentative de déplacement, l'agent vérifie qu'il a bien changé de noeud. Si ce n'est pas le cas, il consulte la position des agents qu'il détecte, si personne ne se trouve dans le noeud qu'il a voulu rejoindre, l'agent conclut que c'est un Wumpus. Dès lors, il envoie la localisation du Wumpus aux agents à proximité. S'il s'avère qu'un agent se trouvait dans le noeud, il envoie la localisation d'un noeud adjacent contenant une odeur s'il y en a.

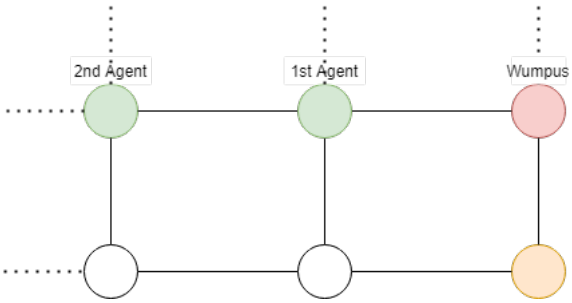

FIGURE 2 – Exemple de situation de blocage 1.

Si l'agent reçoit la localisation d'une stench ou d'un Wumpus, il se dirige vers ce noeud en priorité. Cependant, ce comportement peut créer des problèmes, nous avons noté 2 cas particuliers.
Dans le cas, de la figure 2, avec une portée de 1 pour l'odeur du Wumpus, l'agent 1 détecte le Wumpus et transmet sa localisation à l'agent 2. Ce dernier reçoit la position du Wumpus et va calculer le plus court chemin vers celui-ci. Il va donc essayer de se déplacer vers le noeud de l'agent 1 sans succès. Pour éviter le cas où l'agent 2 essaye infiniment de se déplacer au même endroit, dès lors qu'il n'a pas réussi à bouger, il abandonne la destination reçue par l'agent 1 et effectue son comportement par défaut.

FIGURE 3 – Exemple de situation de blocage 2ta .

Dans le cas de la figure 3, qui est assez similaire à celui de la figure 2, le Wumpus est bloqué par l'agent 1. De ce fait, l'agent 2 qui est bloqué par l'agent 1 devrait abandonner la

destination qu'il a reçue de l'agent 1 et repartir pour explorer l'environnement. Cependant, l'agent 1 continue d'envoyer la position du Wumpus aux agents autour de lui. L'agent 2 va alors recevoir de nouveau la destination et réessayer de se déplacer sur le noeud de l'agent 1. Afin d'éviter cela, l'agent qui partage la localisation d'une stench ou d'un Wumpus dispose d'une liste des agents auxquels il a déjà envoyé l'information. Si un agent est présent dans cette liste, on ne lui renvoie pas l'information.

Pour le cas de la terminaison, nous avons définie un nombre de tentatives n. Si l'agent essaye, consécutivement et sans succès, de se déplacer n fois sur le noeud du Wumpus. Cela signifie que le Wumpus n'a pas bougé pendant les n tentatives, et on considère que le Wumpus est bloqué. Nous sommes conscients que cela repose sur l'hypothèse que le Wumpus se déplace forcément dans un intervalle de temps plus petit que le temps que prend l'agent pour faire n tentatives de déplacement, et donc que ce n'est pas une façon très propre de vérifier la terminaison de l'agent. Nous aurions voulu mettre en place une terminaison dans lequel l'agent consulte sa carte pour examiner les noeuds disponibles autour du Wumpus, et communique avec les autres agents afin de vérifier que chacun de ces noeuds disponibles soit occupé par un agent.

6 Conclusion et Perspectives

Conclusion

L'enjeu de ce projet était d'implémenter un système multi-agent permettant de terminer efficacement une partie de la variante du jeu "Hunt the Wumpus". Il s'agissait donc principalement d'optimiser au plus les phases d'exploration et de chasse par le biais de communications.

Ce projet nous a permis d'avoir une première expérience dans le domaine des systèmes multi-agent et a été pour nous source d'apprentissage. En effet, nous avons pu découvrir comment sont implémentés le comportement des agents par le biais de Behaviours, comment l'initialisation et les transitions entre ceux-ci sont gérés à l'aide d'un FSM et comment les agents communiquent entre eux, notamment par le biais d'une page jaune.

Perspectives

Les comportements implémentés sont fonctionnels et donnent des résultats satisfaisants lors de nos tests. Ceux-ci ont été principalement testés sur une carte grille de taille 5 puis de taille 10, ainsi que sur la carte **map2020-topologyExam1-tree.dgs**, avec 1, 2 et 3 agents.

Néanmoins, plusieurs aspects mentionnés dans le rapport tels que le partage de carte ou la terminaison des agents auraient pu être améliorés. Nous aurions également voulu implémenter des actions jointes entre les agents afin de rendre la chasse du Wumpus plus efficace, et pouvoir travailler sur la distribution du système multi-agent sur différentes machines.