TP - Système-multi agents et architecture BDI

Réalisé par : DEWONE MORCHOISNE et OUSSEYNOU GNING

I – Présentation

On s'intéresse à la modélisation d'agents utilisant une architecture **BDI** (Beliefs, Desire, Intention) dans un environnement dynamique. Pour ce projet nous avons choisi de réaliser une simulation de pompiers dans un environnement présentant des points chauds qui peuvent déclencher des feux aux alentours, feux qui peuvent ensuite se propager. Le projet est réalisé à l'aide du langage de programmation python.

II – Modélisation du système

Pour concevoir un système multi-agents simulant l'intervention de pompiers dans un environnement ouvert et dynamique nous allons utiliser l'architecture **BDI**. Le principe est que des agents patrouillent dans la forêt à la recherche de feux qu'ils doivent éteindre. Pour éteindre ces feux, ils ont accès à de l'eau dans leur base principale au milieu de la carte et doivent la ramener jusqu'au feu avant de pouvoir l'éteindre. Une des complexités de la tâche est le fait que l'environnement est dynamique et que les agents n'ont qu'une connaissance partielle de l'environnement. Les agents n'ont initialement pas de carte de la forêt et doivent remplir cette carte en explorant l'environnement. Il y a dans la forêt des **montagnes** qui représentent des obstacles infranchissables qui ralentissent les agents et des **points chauds** qui sont susceptibles d'enflammer les zones aux alentours (avec une probabilité pe) ce qui aura pour effet de déclencher un feu qui va ensuite pouvoir se propager sur aux cases adjacentes (avec une probabilité pp). Le fait est que les agents ne seront au courant que le feu existe qu'une fois qu'un agent aura découvert ce feu lors de sa patrouille. Si le feu est découvert trop tard, il aura potentiellement pu se propager de manière incontrôlable et il sera alors difficile pour les agents de le contenir. Ils doivent donc patrouiller de manière efficace et organisée. Une des manières de les aider à patrouiller est de les faire communiquer à l'aide de talkie-walkie, ainsi chaque agent est au courant des découvertes des autres agents et ils peuvent d'une part optimiser leurs chemins de patrouille et d'autre part s'entraider pour éteindre les feux. Les agents doivent également se reposer, ils doivent donc régulièrement retourner à la base et se reposer pendant un certain temps.

NB : Une idée était de mettre en place des camions de pompier que les agents pourraient déplacer dans les zones de feux pour réduire le temps de transport de l'eau, le camion aurait alors eu une capacité d'eau à recharger à la base une fois épuisée. Cette idée n'a pu être mise en place par manque de temps.

Ainsi le système peut se réduire à :
- L'ensemble des agents (communique à l'aide de talkie-walkie représenter par une carte partagée)
- l'environnement (Obstacles, Feux, Points chauds, Base)

III – Modélisation du fonctionnement de l'agent

Architecture BDI :

- **Beliefs** : Ensemble des savoirs de l'agent sur son environnement et sur les autres agents, cette base de savoir est mise à jour durant la simulation à chaque fois que l'agent apprend de nouvelles choses. Exemple : il y a une feu à la position (x,y) fait partie de la base de savoir de l'agent si il a découvert ce feu.

Dans notre cas l'agent est au courant de :
- l'énergie qu'il lui reste
- de sa position
- du fait qu'il ait de l'eau sur lui
- des intentions des autres agents
- de l'environnement de manière partielle grâce à une carte qu'il met à jour selon ses découvertes

- **Desires** : Objectifs que l'agent aimerait accomplir, ces objectifs sont stockés comme un ensemble de désirs. La base de désir est mise à jour lorsqu'un désir est comblé ou lorsqu'il est retiré de la base de désirs car plus réalisable selon les croyances de l'agent. Ces désirs sont hiérarchisés selon des relations de sur/sous désirs. Une valeur de priorité de chaque désir est calculée et mis à jour régulièrement, elle va aider lors du choix de l'intention. Exemple : Éteindre le feu à la position (x,y) est un désir.

Dans notre cas l'ensemble des désirs possible de l'agent est :

(- Sur désir → Sous désir → Sous désir etc…)

- Se reposer à la base → Aller à la base
- Explorer → Suivre le chemin path
- Éteindre un feu → Aller vers une case voisine du feu (→ aller chercher de l'eau → aller à la base)

- **Intentions** : Ce que l'agent est entrain d'accomplir. L'intention en cours va déterminer le plan d'action à choisir pour réaliser au mieux cette intention. Ces intentions peuvent être instantanées (prendre de l'eau) ou persistantes (aller à la position (x,y)). Les intentions sont stockées dans des files FIFO et seule la dernière intention est exécutée. L'intention qui va être choisie est l'intention présentant la valeur de priorité la plus élevée.

Dans notre cas l'ensemble des intentions est :

- Aller à la position (x,y), réalisé par l'algorithme de path planning A*
- Se reposer
- Explorer, réalisé avec un algorithme de type RTT (Rapid Random Tree)
- Prendre de l'eau
- Éteindre un feu

Le fonctionnement de notre agent reprend le fonctionnement classique de celui d'un agent BDI bien qu'il y simplifie les différentes fonctions.

Après initialisation des croyances, la **boucle d'action** de l'agent est ainsi la suivante :

- Perception → Mis à jour des croyances
- Mis à jour des désirs et en particulier de leur valeur de priorité en fonctions des croyances
- Choix de l'intention : On garde l'intention courante avec une probabilité p (→persistance) ou sinon on réalise l'intention liée au désir dont la valeur de priorité est la plus élevée (→ réactivité). Une petite valeur de p va engendrer un agent avec une forte persistance mais une faible réactivité et inversement.
- Choix du plan le plus adapté à l'intention en cours à l'aide de la base de croyances.
- Exécution du plan

L'échelle de temps choisi dans notre modèle est celui de l'action, ainsi après chaque action, l'agent va réaliser cette boucle pour choisir son action suivante.

IV – Fonctionnement de quelques fonctions

Pour explorer son environnement, l'agent doit choisir le chemin permettant de maximiser le renouvellement de sa carte. Utiliser un algorithme de **Dijkstra** permettrait de réaliser cette tâche mais cela serait extrêmement chronophage. Pour accélérer le processus, on utilise un algorithme type **RRT**.

Algorithme d'exploration :

L'agent possède une carte partielle de son environnement qui est mis à jour en fonction de ses découvertes mais également une carte de mis à jour des différentes cases. C'est à dire que lorsqu'une case de l'environnement est explorée, elle est mise à jour sur la première carte et son poids est remis à 0 sur la seconde. Une fois que tous les agents ont effectuer leur action, les poids de toutes les cases de la seconde carte sont incrémentés de 1. L'objectif est donc de choisir un chemin qui permet de voir des cases dont le poids est maximum ce qui constitue le score d'exploration. Pour cela, au départ, on choisit parmi 2 cases voisines de l'emplacement de l'agent celle qui maximise le score d'exploration et une autre case aléatoire. Ces 2 nouvelles cases vont être 2 nouveaux nœuds à explorer. On répète l'opération précédente sur nos 2 nouveaux nœuds ce qui va créer en tout 4 chemins. Et on répète l'opération 10 fois jusqu'à obtenir des chemins de taille 10. Le chemin sélectionner est celui qui maximise le score d'exploration.

Pour trouver un chemin permettant à l'agent de se déplacer de sa case de départ à la case d'arrivée tout en prenant en compte les obstacles on utilise l'algorithme **A***. Il est beaucoup plus rapide que l'algorithme de Dijkstra et donne des résultats très satisfaisant. De plus, du fait que le calcul est réactualisé à chaque pas, ces 2 algorithmes devraient donner le même résultat.

V – Résultats

Les premiers résultats du projet peuvent être observer à l'aide de la sommaire interface graphique réalisée. La plupart des méthodes fonctionnes mais il reste cependant pas mal cas à étudier et à debugger ce qui n'a pas été réalisé par manque de temps. On peut notamment voir que la mise à jour des positions des agents pose problème ainsi que l'utilisation de l'algorithme A*