Création d'un agent aspirateur

Yoann Ayoub *AYOY01080003*, Louise Lizé *LIZL16580007* et Abdou Dieng *DIEA.....*

3 octobre 2022

Résumé

Ce document présente le travail réalisé afin de créer un agent aspirateur capable d'effectuer différentes tâches dans un environnement donné. Nous présenterons notre modélisation du problème et montrerons la conformité des différents éléments avec l'architecture présentée en cours. Enfin, ce rapport nous permettra également de positionner notre implémentation en décrivant le type d'environnement (propriétés), le type d'agent, la fonction d'agent, modélisation de l'action, de la perception, etc.

1 Modélisation du problème

Notre problème se décompose en deux éléments, d'un côté le robot aspirateur et de l'autre l'environnement dans lequel il évolue.

1.1 L'environnement

L'environnement est une carte contenant 5*5 = 25 cases. Il s'exécute dans une boucle infinie et génère des poussières et des bijoux de façon aléatoire sur la carte. Il est :

- Complètement observable
- Stochastique, le prochain état de l'environnement ne dépend pas de son état courant
- Épisodique, le prochain évènement ne dépend pas des précédents
- Dynamique, la carte de l'environnement peut changer pendant le cycle de vie de l'agent aspirateur
- Discret, l'environnement possède un nombre fini d'actions, de perceptions et d'états possibles

1.2 L'agent

L'agent aspirateur est un agent basé sur les buts. Il peut être formulé par un problème à simple état :

- Etat initial : Le robot est positionné en (2,2) soit au centre de la carte, et à l'état initial « Ne rien faire ».
- Opérateurs : Ne *Rien* faire, Aller à *Gauche*, Aller à *Droite*, Aller vers le *Haut*, Aller vers le *Bas*, *Aspirer* une poussière, *Ramasser* un bijou
- Fonction de succession {Action, État} : {Droite, le robot aspirateur se déplace d'une case vers la droite}, {Gauche, le robot aspirateur se déplace d'une case vers la gauche}, {Haut, le robot aspirateur se déplace d'une case vers le haut}, {Bas, le robot aspirateur se déplace d'une case vers le bas}, {Aspirer, le robot aspirateur aspire la poussière et met à jour ses croyances}, {Ramasser, le robot aspirateur ramasse le bijou et met à jour ses croyances}, {Rien, le robot aspirateur attend}
- Test de but : rien ne se trouve sur la carte (ni poussière, ni bijou)
- Coût : 1 unité par action

1.3 Conformité avec l'architecture vu en cours

Afin de nous conformer à l'architecture vue en cours, l'environnement et le robot aspirateur seront chacun sur un thread différent. De plus, le robot sera basé sur les buts selon un fonctionnement BDI (Beliefs, Desire, Intentions) qui sera détaillé plus tard. Nous nous sommes également basés sur le cours pour créer les algorithmes d'exploration non informé et informé (ainsi que l'heuristique correspondante). Enfin l'ensemble des développements, qu'il s'agisse de l'environnement ou du robot (excepté pour l'exploration informée) a été réalisé de manière générique de sorte à ce que l'on puisse facilement modifié le problème en agrandissant la carte, ajoutant un type d'objet ou un type d'action, etc.

2 Description de l'implémentation

Nous allons maintenant décrire l'implémentation que nous avons mise en place conformément à la modélisation du problème et l'architecture vue précédemment.

2.1 Environnement et propriétés

Afin de respecter la modélisation décrite, tout en ayant un environnement le plus générique possible nous avons utilisé deux classes :

- la première classe est une matrice représentant la carte, elle est notamment utilisée pour choisir les dimensions de la map
- la seconde représente une case. Nous avons choisi d'utiliser un dictionnaire pour chaque case avec la possibilité de contenir, ou non, différents éléments (bijou, poussière et robot)

Ainsi, la première classe fait appel à la seconde pour générer la carte. L'intérêt de cette modélisation et que si l'on souhaite modifier le problème, en ajoutant un type d'objet par exemple, on peut le faire facilement et sans faire exploser la combinatoire (contrairement à une modélisation avec des chiffres dans une matrice).

Concernant la génération aléatoire de bijou et de poussière, elle se fait selon une probabilité fixée pour chaque objet. L'environnement commence par choisir aléatoirement l'objet qu'il souhaite générer, puis il choisit aléatoirement une case de la carte et vérifie que l'objet ne s'y trouve pas déjà. Dans quel cas il choisit aléatoirement une nouvelle case jusqu'à ce qu'il trouve une case sur laquelle ne se trouve pas l'objet.

2.2 Type d'agent et propriétés

2.2.1 Fonctionnement de l'agent

Tant que l'agent est en vie, il se trouve dans une boucle infinie dans laquelle il effectue les étapes :

- Observation de l'environnement. L'agent demande à l'environnement son état à l'instant et affecte la réponse à ses croyances.
- Test de désir. L'agent compare ses croyances à son objectif, à savoir une carte de l'environnement vide, sans aucun objet dessus.
- Affectation d'une série d'actions à ses intentions. Quand les croyances de l'agent sont différentes de son but, l'agent fait appel à un algorithme d'exploration (détaillé par la suite) pour récupérer une série d'action qu'il affecte à ses intentions. Sinon il ne fait rien.
- Exécution des intentions. L'agent effectue les actions présentes dans ces intentions et met à jour ses croyances au fur et à mesure.

2.2.2 Modélisation des actions

Concernant la modélisation des actions elle se fait tel que décrit dans le tableau 1.

TABLE 1 – Prémisses et Conséquences en fonction de chaque Action

Action	Prémisse	Conséquence
Ramasse	case avec bijou & agent	bijou ramassé
Aspire	case avec poussière & agent	poussière aspirée
Se déplace	case suivante de la carte	agent déplacé
Rien faire	carte sans aucun objet	agent attend

2.2.3 Modélisation de la perception

L'agent perçoit son environnement en observant une carte de l'environnement à un moment donné. Il affecte cette observation à ses croyances et les utilise ensuite pour explorer la carte et agir sur les objets (poussières et bijoux).

2.2.4 BDI

L'agent peut être dans un état *propre* (carte sans objet) ou *salle* (carte contient au moins un objet) en fonction de son observation de l'environnement et donc de ses croyances.

De par la méthode BDI, l'agent possède en tout temps trois attributs :

- Beliefs : les croyances de l'agent sont données par la dernière carte dont il a eu connaissance
- Desire : le désir de l'agent est d'atteindre son but, à savoir que la carte soit vide et ne contienne aucun objet. Il utilise ses désirs en les comparant à ses croyances afin d'évaluer les tâches qu'il doit effectuer
- Intentions : les intentions de l'agent sont issues du résultat de l'exploration. Il s'agit d'une liste d'action à effectuer pour atteindre son but

2.3 L'exploration

Maintenant que les états et le fonctionnement de l'agent ont été expliqués, nous allons détailler le fonc-

tionnement des algorithmes d'exploration auxquels le robot fait appel pour déterminer ces intentions.

Deux types d'exploration ont été mis en place. La première est une exploration non-informée générique de type breadth first search, la seconde est une exploration informée de type greedy search avec pour heuristique l'utilisation de la distance de Manhattan.

2.3.1 Non Informée

L'exploration non informée en largeur a été implémentée par une liste de liste. Celle-ci ne représente pas vraiment l'arbre, mais les différents chemins de ce dernier.

Initialement, le seul élément de la liste est la position du robot au moment où il commence son exploration, on a donc une liste avec pour seule valeur la position initiale [[(x,y)]].

Puis à la deuxième itération on remplace cette première liste par 4 listes avec pour chacune, la position initiale et une case voisine. Ces 4 listes ont alors chacune deux éléments. On a alors une liste composé des listes [[(x,y),(x+1,y)],[(x,y),(x-1,y)],[(x,y),(x,y+1)],[(x,y),(x,y-1)]].

Et ainsi de suite, à chaque itération on remplace chaque liste par 2 à 4 listes (en fonction du nombre de voisins) contenants le chemin tracé dans la liste précédente et à laquelle on ajoute un voisin de la dernière coordonnées.

A chaque itération on vérifie s'il existe une liste qui contient l'ensemble des coordonnées par lesquelles le robot sait qu'il doit passer. Ces coordonnées sont les coordonnées de tous les bijoux et poussières dont le robot à conscience de par son observation dans ses croyances. Ainsi, dès que cette condition est vérifiée, l'exploration s'arrête et cette liste est renvoyée au robot. Elle contient donc le chemin qu'il doit parcourir pour passer par l'ensemble de ces buts de sorte à ce qu'il puisse effectuer les actions nécessaires pour chacun.

2.3.2 Informée

Pour l'exploration informée, nous utilisons la même logique par liste que pour l'exploration non informée. Sauf qu'au lieu de développer l'ensemble des branches de l'arbre, nous utilisons un greedy search avec l'heuristique des distances de Manhattan pour définir le point duquel on souhaite se rapprocher et ne développer que les branches de l'arbre correspondantes.

L'heuristique commence par calculer la distance de Manhattan entre le robot et chaque objet de la carte. L'agent défini comme but l'objet dont il est le plus proche. Dans le cas où deux objets sont à la même distance de Manhattan du robot, ce dernier choisi aléatoirement son but.

Ainsi, l'agent utilise l'heuristique pour définir son but. Cependant afin de respecter la modélisation du problème nous souhaitons que l'exploration fournisse à l'agent un plan lui permettant de traiter l'ensemble des objets dont il a conscience sur la carte.

C'est pourquoi une fois que l'algorithme donne le chemin jusqu'à l'objet le plus proche, on retire cet objet de la liste et on relance l'exploration à partir des coordonnées de ce premier objet.

Ainsi, on obtient le chemin jusqu'au deuxième objet et on ajoute ce chemin au premier. On procède ainsi de suite jusqu'à ce que l'ensemble des objets de la carte aient été traités. On peut alors renvoyer la série d'actions aux intentions de l'agent pour que ce dernier les effectue.

Ici l'exploration ne fournit pas forcément le meilleur chemin, au sens du chemin le plus économique énergétiquement. Effectivement, dans le cas où deux objets sont à la même distance on choisit aléatoirement celui vers lequel on se dirige. Cependant l'exemple de la Figure 1 montre que ce choix aléatoire peut mener à un choix non optimal.

FIGURE 1 – Exemple de carte : Robot - Jewel - Dust

Ici si l'on commence par le Jewel de gauche, alors il faudra 10 coups pour traiter tous les objets au lieu de 8 si l'on commence par le Jewel de droite.

Afin de s'assurer d'avoir un chemin optimal il aurait fallu légèrement modifier l'heuristique. Plutôt que de choisir aléatoirement l'objet en cas de distance égale, il faudrait prendre celui qui est le plus éloigné de l'ensemble des autres objets. Cela peut se traduire par le fait d'associer à chaque objet une valeur égale à la somme des distances de Manhattan qui le sépare des autres objet. Dans notre exemple le Jewel de gauche aurait une valeur égale à 2+4=6 et celui de droite 2+6=8.

Ainsi en cas d'égalité, on choisi l'objet pour laquelle

cette valeur est la plus grande (le plus éloigné de l'ensemble des autres objets) ce qui assure l'optimalité de l'algorithme.

2.4 Apprentissage et performance

Enfin , une fois l'exploration non informée et informée utilisées, on commence l'apprentissage. L'objectif de cet apprentissage est de déterminer la fréquence d'observation qui permet d'obtenir la meilleure performance.

2.4.1 Mesure de performance

La mesure de performance se fait tel que sur un temps donné, on calcule le nombre de cases propres, auxquelles on soustrait la consommation électrique de l'agent et les pénalités (en cas d'aspiration d'un bijou par exemple).

2.4.2 Apprentissage

L'apprentissage à donc pour objectif de trouver la fréquence d'observation permettant de maximiser la performance.

Pour ce faire nous avons mesurer la performance du robot sur une minute en fonction du nombre d'actions i qu'il réalise avant de ré-observer son environnement.

Pour i=1, l'agent n'effectue que la première action de son plan avant d'observer à nouveau et de recalculer son plan. Pour i=2, les deux premières et ainsi de suite.

Nous mesurons donc la performance de l'agent pour chaque valeur de i puis une fois que l'ensemble des mesures ont été réalisées, nous gardons la valeur de i pour laquelle la performance est la plus importante.

Ce qui constitue un apprentissage de la meilleure fréquence d'observation.