Projet: Search and Rescue

Cédric Buron

14 février 2024

Introduction

Le projet consiste à concevoir une équipe de robots dans le contexte d'une mission de "Search and Rescue". Plusieurs types de robots, avec des capacités de déplacement, de communication et de perception seront mis à votre disposition. Votre objectif est de déterminer la stratégie de ces robots, et donc de concevoir leur comportement, mais aussi de déterminer la composition de votre équipe afin de remplir la mission le plus vite possible. L'environnement est constitué d'un labyrinthe avec des murs infranchissables. Dans ce labyrinthe se trouve une personne en danger et deux objets permettant de la secourir. Le problème est que les robots ne connaissent pas la localisation de ces éléments. Le plan du labyrinthe est, lui, connu. Le projet est conçu en python, avec le module Mesa. L'interface grapphique est très simple, mais elle vous permettra de suivre en temps réel ce qui se passe durant la mission. Afin de vous épargner des détails d'implémentation parfois complexes à mettre en œuvre, je vous ai fourni un certain nombre de primitives, que vous pourrez utiliser durant le projet.

Il est important de noter que bien qu'il s'agisse d'une compétition **vous ne serez pas noté·es principalement sur les performances de votre équipe**. Le temps imparti est trop court pour que vous puissiez vraiment mettre en place une politique d'amélioration de votre équipe. Un choix peut paraître pertinent et finalement ne pas s'avérer payant. Vous serez, en revanche, évalué·es sur la pertinence de vos choix.

Objectif

Le projet se présente ainsi : un labyrinthe (vous aurez un labyrinthe pour vous exercer, mais un labyrinthe différent servira à l'évaluation) est généré. Dans ce labyrinthe sont générés deux types d'agents inertes (sans aucun comportement) : un agent de type `Person` et deux de type `RescueItem`. L'agent de type `Person` ne peux être déplacé. Les agents de type `RescueItem` peuvent être déplacés par certains robots qui ont les primitives appropriées (voir ci-dessous). L'objectif du projet est de faire récupérer les deux `RescueItem` par les agents qui en ont la capacité et de les amener jusqu'à l'agent `Person` où il faudra les déposer.

Robots et primitives

Les types de robots sont décrits ci-après, mais tous héritent de la classe `Robot`. Cette classe hérite elle-même de la classe `mesa.Agent`. Toutes les définitions sont données dans le module `projet_import`. Vous pouvez, si vous le souhaitez, voir comment cela est implémenté, mais cela n'est pas obligatoire, et le cas d'application m'a imposé de faire pas mal de modifications ad hoc, pas toujours très compréhensibles lorsqu'on n'a pas été confronté·e au problème. Cependant, si cela vous intéresse, n'hésitez pas à jeter un œil et à me poser les questions qui vous paraîtront pertinentes.

Votre objectif sera d'étendre les sous-classes de `Robot` (dont les caractéristiques sont données ci-dessous) et de définir leur fonction `step` qui leur donnera leur comportement. Pour cela, vous pourrez utiliser un certain nombre de fonctions définies dans la classe `Robot`.

Percevoir

Les robots sont capables de perception. À chaque tour, le robot peut percevoir son environnement, et détecter si un `Item` (une `Person` ou un `RescueItem`) est visible au moyen de la méthode `def sense(self) -> List[Item]`. La liste d'items récupérée peut être stockée dans une variable locale si vous ne souhaitez pas l'oublier.

Se déplacer

Pour vous éviter d'avoir à concevoir la planification de trajectoire, qui sort complètement de l'objectif de ce cours, une fonction vous a été fournie : `def goto(self, destination_x, destination_y)` Cette méthode va déplacer le robot vers le point qui lui est donné en destination, et s'il l'atteint, s'y arrêter.

⚠ Attention Vous devez absolument utiliser cette méthode porur vous déplacer. Vous n'avez pas le droit de modifier vos `x` et `y` vous-même. De même, vous n'avez le droit d'utiliser qu'un `goto` par `step`.

Selon les types de robots, la méthode `goto` est surchargée différemment. Pour les agents devant se déplacer selon les murs du labyrinthe, le calcul de la trajectoire est basé sur l'algorithme de Dijkstra.

Communiquer

Comme dans le TP 2, une boîte de messagerie a été implémentée. Les `Robot` sont dotés de deux primitives :

- une primitive `def send(self, msg) -> None` qui permet d'envoyer un message de type `Message` (cette classe est identique à celle qui vous est donnée dans le TP2) à un agent ou une liste d'agent. **Attention cependant** le message ne sera envoyé qu'aux agents qui se trouvent à portée ;
- et une primitive `def receive(self) -> List[Message]:` qui permet de récupérer l'ensemble des messages de la boîte mail **et de la vider**.

Manipuler les objets

Il est enfin possible pour certains types de robots de manipuler les `RescueItem`. Ces robots sont dotés de deux primitives supplémentaires : `def take(self, item: RescueItem)` qui permet de récupérer un `RescueItem` dont on partage la localisation et `def drop_item(self)` qui permet de poser un `RescueItem` qu'on a récupéré. **Attention, il n'est pas possible de transporter plus d'un `RescueItem` à la fois**. Récupérer un `RescueItem` alors qu'on en tient déjà un fera forcément tomber l'autre.