Alan GUIVARCH
GUIA24119800

Sacha GUYOT
GUYS07119908

INF846
Intelligence artificielle
-
TP1

2020-2021

1. Modélisation de l'environnement

Pour modéliser l'environnement nous avons créé une classe Environment qui extends la classe Thread pour pouvoir faire fonctionner l'environnement dans un fil d'exécution différent de celui de l'acteur.

La carte du manoir est représentée par l'attribut `map`, une matrice d'int. Les valeurs de cette matrice sont définies par un bitmask. Le bit 0 signifie qu'il y a de la poussière dans la pièce et le bit 1 qu'un bijou est tombé à cet endroit. 0 signifie donc que la pièce est propre et sans bijoux, 1 que la pièce est poussiéreuse, 2 que la pièce contient un bijou et 3 que la pièce est sale et contient un bijou.
A chaque boucle de l'environnement, il y a une certaine probabilité que de la poussière et des bijoux soient générés sur la carte du manoir. Ils sont générés grâce à la fonction `generate`.

Notre environnement est :

- Complètement observable
- Stochastique (l'état change en fonction des actions du robot mais aussi lorsque de la poussière ou des bijoux sont générés)
- Épisodique (les décisions ne dépendent pas des décisions précédentes)
- Dynamique (l'environnement peut ajouter de la poussière ou des bijoux pendant que le robot réfléchit)
- Discret (le nombre d'états possibles est gigantesque mais fini)
- Mono-agent

2. Modélisation de l'agent

Pour modéliser l'agent nous avons créé une classe Aspirobot qui extends la classe Thread pour pouvoir faire fonctionner l'agent dans un fil d'exécution différent de celui de l'environnement.

Notre agent est basé sur les buts. L'état mental BDI de l'agent est contenu dans la classe et représenté par les objets suivants :

- `beliefs` est une matrice à 2 dimensions représentant le manoir. Il s'agit d'une copie de `map` mise à jour à chaque appel aux capteurs.
- La fonction `desires` vérifie si les désirs de l'agent sont vérifiés, c'est à dire que toutes les pièces du manoir sont propres et qu'aucun bijou ne traine par terre.
- Les `intentions` de l'agent sont données par les fonctions `decideN_W`, `decideN_D` et `decideI_BF` (trois types de recherche : non informée en

largeur, non informée en profondeur et informée best first). Les `intentions` sont représentées par une liste de caractères représentant chacun une action à effectuer.

Pour modéliser les capteurs et effecteurs de l'agent nous avons créé une classe `Sensors` et une classe `Effectors`.
Le rôle de `Sensors` est de mettre à jours l'attribut `beliefs` de l'agent en copiant la carte du manoir contenue dans l'environnement. Elle permet aussi au robot de connaître son score.
Le rôle de `Effectors` est de réaliser les actions demandées par l'Aspirobot :

- la fonction `suck` permet d'aspirer la poussière sur la case où se trouve le robot. Elle aspire aussi l'éventuel bijou qui pourrait s'y trouver.
- la fonction `move` permet au robot de bouger dans la direction donnée en argument.
- la fonction `pick` permet au robot de récupérer le bijou qui se trouve sur sa case.

Dans ces trois cas, l'énergie dépensée par le robot est prise en compte par l'attribut `score` de l'environnement, qui est incrémenté du coût de l'électricité défini dans `Constants`.
Si l'Aspirobot aspire un bijou, il subit une pénalité de score dont la valeur est également définie dans `Constants`.
Plus le score du robot est élevé plus ses performances sont mauvaises.

3. Les arguments

Nous pouvons modifier la taille du manoir en utilisant les arguments `"h <int>"` pour la hauteur et `"w <int>"` pour la largeur.

Nous pouvons modifier les probabilités d'apparition des bijoux et de la poussière avec `"d <double>"` pour la poussière et `"j <double>"` pour les bijoux. Les probabilités doivent être comprises entre 0 et 1.

Nous pouvons également modifier le coût de l'électricité (la pénalité de score appliquée dès que le robot fait appel à ses effecteurs) avec `"ec <double>"` et la pénalité subie lorsqu'il aspire un bijou avec `"jc <double>"`.

Les paramètres `db <double>` et `jb <double>` permettent de régler les bonus de score accordés au robot lorsqu'il aspire de la poussière et ramasse un bijou respectivement.

Ensuite nous pouvons modifier le mode de recherche de l'Aspirobot avec `"mode <string>"`. Les trois modes possibles sont :

- `n_w` pour l'exploration non informée en largeur
- `n_d` pour l'exploration non informée en profondeur
- `i_bf` pour l'exploration informée best first

Nous pouvons décider d'afficher la carte du manoir avec l'argument `"showmap yes"` ou de ne pas l'afficher avec `"showmap no"`.

Enfin nous pouvons modifier la durée de chaque test en utilisant `"time <int>"` et le nombre de tests effectués pour chaque limite avec `"runs <int>"`. Un test correspond à un lancement de la simulation avec une limite de nombre d'actions à effectuer entre deux explorations. Le robot peut soit effectuer <limite> actions puis refaire une exploration, soit effectuer toutes les actions prévues puis refaire une exploration, si le nombre d'actions retournées par la fonction de décision est inférieur à la limite.

Valeurs par défaut :

- `h = 5`
- `w = 5`
- `d = 0.15`
- `j = 0.05`
- `ec = 1`
- `jc = 1000`
- `db = 5`
- `jb = 2`
- `mode = "n_w"`
- `showmap = "yes"`
- `time = 10`
- `runs = 10`

L'heuristique pour l'exploration informée consiste simplement à compter le nombre de pièces poussiéreuses et le nombre de pièces contenant un bijou et à additionner ces nombres. Il est possible de donner une pondération à ces nombres en utilisant les paramètres `ds <double>` et `js <double>` qui règlent respectivement la pondération de la poussière et des bijoux. Les pondérations par défaut valent 1.