Fiche TP01

marc-michel.corsini@u-bordeaux.fr

9 février 2016

Pour ce TP (qui doit être terminé pour la semaine après les vacances de février) nous allons aborder la mise en place d'un aspirateur fonctionnant sur des règles.

1 Base de connaissances et règles

Le fichier **briques.py** contient deux classes, la classe `Rule` qui définit la notion de règle et la classe `KB` qui définit la notion de base de connaissances.

1.1 Rule

Une règle est constituée d'une partie gauche : « la condition » qui définit quand est-ce que la règle est appliquée, et une partie droite contenant l'action à utiliser.

- La condition est définie par les perceptions de l'agent données sous forme d'une liste
- L'action est une des actions autorisées pour l'agent

Un objet de la classe `Rule` dispose de différents attributs :

- **condition** qui renvoie les perceptions nécessaires pour son utilisation
- **conclusion** qui renvoie l'action à effectuer
- **nbUsage** qui renvoie le nombre de fois où cette règle a été utilisée
- **scoreTotal** qui renvoie la somme des récompenses reçues suite à l'utilisation de cette règle
- **scoreMoyen** qui renvoie le ration scoreTotal / nbUsage

Le constructeur de la classe prend en entrée : une liste de percepts, une action et un score

1.2 KB

La base de connaissances est en fait un dictionnaire de Rule, indexée par la partie condition de la règle.

- **eraseBase()** réinitialise à vide la base
- **deletePercept(percept)** détruit toutes les règles de la base ayant pour condition `percept`
- **find(percept)** renvoie la liste des règles (Rule) de la base ayant `percept` comme condition
- **add(rule)** ajoute la règle **rule** dans la base, et fait la mise à jour des valeurs nbUsage, scoreTotal et scoreMoyen.

2 Capteurs

Puisque nos aspirateurs évoluent dans un monde en deux dimensions, on peut adjoindre, au maximum 9 capteurs (numérotés de 0 à 8).

7	0	1
6	8	2
5	4	3

On suppose, pour le moment que notre aspirateur regarde toujours au nord de la carte. Si on désigne par N, E, S et W les points cardinaux, les capteurs correspondent donc à "N NE E SE S SW W NW ICI".split()

```
>>> l = "N NE E SE S SW W NW ICI".split()
>>> for i in range(9): print(i,l[i])
...
0 N
1 NE
2 E
3 SE
4 S
5 SW
6 W
7 NW
8 ICI
```

3 Le but de ce TP

Mettre en place la comparaison de différents aspirateurs

1. Un aspirateur aléatoire
2. Un aspirateur dont les règles sont écrites par vous (et qui sont figées)
3. Un aspirateur dont les règles sont trouvées par expérience du monde

Pour pouvoir fonctionner, il faut que vous ayez fini le TD2 (test_tp00a). La classe `Aspirateur` utilisée avec la classe `Monde` correspond presque à ce que l'on veut pour faire tourner l'aspirateur aléatoire. Il ne manque en fait que le réglage des évaluations – `getEvaluation()` de **Aspirateur** et `perfGlobale` de **Monde**.

3.1 Evaluations

On impose que **getEvaluation()** soit définie comme

$$\frac{1}{\text{taille base} + 1} \times (\text{nombre de pièces nettoyées} + 1)$$

On impose que **perfGlobale** soit définie comme

$$\text{nombre de pièces nettoyées} - \text{nombre de pièces visitées 3 fois ou plus}$$

3.2 Simulations

Chaque aspirateur sera testé 10 fois dans un monde de taille 1×2 pendant 4 itérations, puis 10 fois dans un monde de taille 1×5 pendant 10 itérations. **Attention** l'aspirateur ne sait pas où il se trouve.

1. Aspirateur aléatoire aura une base vide et pas de capteurs
2. Aspirateur avec règles prédéfinies, deux versions seront testées, l'une avec capteur ICI, l'autre avec capteurs Est et ICI. Cet aspirateur n'apprend pas.
3. Aspirateur apprenant en trois versions un seul capteur ICI, deux capteurs Est et ICI, trois capteurs Est ICI et West

3.3 World, une extension de Monde

La classe `World` est dérivée de la classe `Monde`. Il faudra redéfinir les méthodes et attributs

1. **getPerceptions**
2. **applyChoix**
3. **perfGlobale**

3.4 Aspirateur_KB, une extension de Aspirateur

Il y aura en fait deux versions de cette classe, dans l'une le système apprend, dans l'autre le système n'apprend pas de nouvelles règles. Il faudra redéfinir les méthodes et créer trois attributs

1. **knowledge** attribut en lecture/écriture (KB)
2. **apprentissage** attribut en lecture/écriture (bool)
3. **probaExploitation** attribut en lecture seule (float dans [0,1])
4. **getEvaluation**
5. **getDecision**
6. **setReward**

4 Travail à effectuer

Vous récupèrerez le fichier briques.py et le fichier tp01.py que vous complèterez en vous aidant de cette fiche et des commentaires dans les parties manquantes (algorithme qu'il faudra traduire en python).
Oui, je sais ! ce n'est pas tout à fait ce que j'ai raconté en cours.