Fiche TP00

marc-michel.corsini@u-bordeaux.fr

3 février 2016

Ce premier TP (2 séances) vise à mettre en place les éléments de base pour la simulation d'un aspirateur sans capteur, dans un monde de deux cases, chaque case correspondant, ici, à une pièce.

On définit 3 entités pour effectuer les interactions :

- Un dictionnaire `objetsStatiques` dont les clefs sont numériques et dont les éléments sont des paires constituées d'une chaîne de caractères décrivant l'objet et d'un caractère servant à l'affichage. La clef **100** est réservée pour l'aspirateur, la clef **0** pour le « propre »(rien) et la clef **1** pour le « sale » ou la poussière.
- La classe `Monde` prend en premier paramètre un agent Aspirateur, le second paramètre est le nombre de lignes (par défaut 1), le troisième est le nombre de colonnes (par défaut 2). Le monde est donc un ensemble de cases organisées en ligne et colonne, chaque ligne possède le même nombre de cases. Une case ne peut contenir qu'un objet (une clef de `objetsStatiques`).
- La classe `Aspirateur` prend deux paramètres, le premier est une liste des cases voisines visibles par l'agent, par défaut elle est vide. Le second est une liste de chaîne de caractères représentant les actions possibles pour l'agent, par défaut il y a trois actions « Gauche », « Droite », « Aspirer »

```
objetsStatiques = {100: ('Aspirateur','@'),
                   0: ('nothing','.'),
                   1: ('poussière','x'),
               }

class Aspirateur(object):
    """ classe de base pour l'agent mobile """
    def __init__(self, capteurs=[], actions=['Gauche', 'Droite', 'Aspirer']):


class Monde(object):
    """ l'environnement et les interactions avec l'agent """
    def __init__(self,agent,nbl=1,nbc=2):
```

1 Attributs

Tous les attributs que nous utilisons sont en **lecture seule**.

1.1 Aspirateur

- `capteurs` renvoie la liste des capteurs disponibles pour l'agent aspirateur
- `actions` renvoie la liste des actions disponibles pour l'agent Aspirateur
- `vivant` renvoie une valeur **booléenne** True ou False, indiquant si l'agent est « vivant »c'est-à-dire apte à prendre une décision.

1.2 Monde

- `table` est une liste de listes. La longueur de la liste est le nombre de lignes du monde 2D. Les listes intérieures sont de longueur le nombre de colonnes du monde 2D. Les valeurs appartiennent aux clefs de `objetsStatiques` à l'exclusion de la clef 100
- `posAgent` est une paire d'entiers, le premier est un numéro de ligne, le second un numéro de colonne. Cet attribut permet de connaître la position de l'agent dans le monde
- `agent` permet de savoir quel est l'agent qui agit dans le monde
- `historique` est une liste dont les éléments sont des paires, le premier est l'information sur l'état du monde c'est-à-dire la table et la position de l'agent, le second est l'action qu'a effectué l'agent. La position dans la liste est le « moment »où l'événement a eu lieu

  ```
  [ (([ [0,1] ], (0,0)), "Aspirer"), (([ [0,1] ], (0,0)), "Droite"),
     (([ [0,1] ], (0,1)), "Aspirer"), (([ [0,0] ], (0,1)), "Droite") ]
  ```

  Cet historique indique qu'au départ il y avait une case propre et une case sale, que l'agent était dans la case la plus à gauche, qu'il a effectué d'abord Aspirer, puis Droite, puis Aspirer puis Droite.
- `perfGlobale` renvoie une valeur numérique qui est une évaluation de la qualité de la résolution de la tâche par l'agent dans le monde. Comment est calculée cette valeur sera à votre charge dans la suite du projet. On souhaite juste que l'évaluation reflète que l'on souhaite un « nettoyage rapide et efficace du logement ».

2 Méthodes

Pour chaque méthode on fournit la signature et son descriptif succinct. **None** est un mot clef du langage python, qui symbolise l'absence de valeur

2.1 Aspirateur

1. `setReward` : Aspirateur $\times \mathbb{R} \rightarrow$ { None }
   Récupère (et stocke) une information numérique instantannée :
2. `getEvaluation` : Aspirateur $\rightarrow \mathbb{R}$
   Renvoie la performance de l'aspirateur à la fin d'une expérience (n'est pas forcément la même évaluation que `perfGlobale`) ;
3. `getDecision` : Aspirateur $\times$ Liste de clefs $\rightarrow$ actions
   Prend une liste de clefs appartenant à `objetsStatiques` correspondant à ce que l'agent a perçu du monde via ses capteurs et renvoie l'action qu'il veut effectuer.

2.2 Monde

1. `initialisation` : Monde $\rightarrow$ { None }
   Initialise `table`, `posAgent`, `historique`
2. `__str__` : Monde $\rightarrow$ String
   Renvoie une chaine de caractères imprimables représentant le monde et l'agent dans le monde ;
3. `getPerception` : Monde $\times$ Liste de cases $\rightarrow$ Liste d'objets du monde
   Renvoie une liste de taille identique à celle fournie en entrée, et dont les valeurs sont des entiers correspondant aux objets présents dans le monde à ces endroits là. Cette méthode va permettre de fournir à l'agent ce qu'il perçoit via ses capteurs ;
4. `applyChoix` : Monde $\times$ actions $\rightarrow \mathbb{R}$
   Applique les effets de l'action sur le monde, cela modifie `table` et `posAgent`. Renvoie une évaluation de l'intérêt de cette action (négatif = mauvais, positif = bon, 0 = sans effet) ;

5. `updateWorld` : Monde → { None }
   Applique les règles du monde, cela a éventuellement un effet sur `table`. Permet de simuler un monde stochastique.

6. `step` : Monde → { None }
   Effectue un pas de simulation, fait appel successivement à `self.getPerception, self.agent.getDecision, self.applyChoix, self.updateWorld`.

```
    def step(self):
     """ une étape de calcul c'est
(a) connaitre les besoins de l'agent aka perception
(b) demander a l'agent son choix en fonction de ses percepts
(b') sauvegarder l'état du monde au moment de l'action
(c) mettre à jour le monde et envoyer un feedback à l'agent
(d) mettre à jour le monde pour les evts stochastiques
"""
```

7. `simulation` : Monde $\times \mathbb{N} \rightarrow \mathbb{R}$
   Reçoit en entrée le nombre maximum d'itérations, tant que ce nombre n'est pas atteint et que l'agent est vivant, appeler `step`. À la fin du calcul renvoyer la quantité `perfGlobale`

3 Travail à effectuer

- Il faut que le programme fonctionne dans le monde par défaut 1×2, monde dynamique déterministe d'un agent sans capteurs prenant ses décisions aléatoirement sans apprentissage (sans feedback). Les tests doivent être réussis à 100%

- Vous pouvez ensuite tester le programme pour un aspirateur percevant la case sur laquelle il est. Le paramètre « capteurs »sera alors la liste **[8]**. Pour cela vous créerez une classe **AspiClairvoyant** dérivée de Aspirateur pour laquelle vous n'aurez qu'à changer la méthode getDecision. Cet aspirateur connait la physique du monde (il sait donc que la saleté peut être aspirée, mais s'il connait le nom décrivant la saleté, s'il sait qu'il faut regarder dans le dictionnaire `objetsStatiques`, il ne sait pas qu'elle est la clef associée)

- **Pour ceux qui ont fini** réfléchissez/essayez de mettre en place une classe **AspiVoyant** qui apprenne à aspirer la saleté et uniquement la saleté.