TP sur le *Q-Learning*
Cours APR, M2 Info

F. Pesquerel & Ph. Preux

22 novembre 2021

1 Préliminaires

L'objectif de ce TP est de prendre contact avec le *Q-Learning* tabulaire, algorithme 4 du poly. Pour cela, on va expérimenter avec une tâche de sortie de labyrinthe.

2 Prise en main de l'environnement d'expérimentation

`labyrinthe.py` est un environnement gym qui permet de résoudre des problèmes de labyrinthe : l'agent doit trouver la sortie d'un labyrinthe le plus vite possible. Le plan du labyrinthe est fourni dans un fichier texte très simple. Par exemple, le fichier `laby.1` contient :

```
*******
* *   *
* * * *
* *** *
*  *  *
* *** *
* * * *
*     G
*******
```

Les `*` indiquent les murs, les espaces indiquent les couloirs, le `G` indique la case à atteindre. Chaque case est repérée par ses coordonnées cartésiennes :

— l'abcisse d'une case est numérotée à partir de 0 pour la case la plus à gauche

— l'ordonnée d'une case est numérotée à partir de 0, de haut en bas.

La case `G` a donc les coordonnées $(6, 7)$.

Le retour est défini comme suit : 10 quand on atteint la sortie, 0 sinon.

2.1 Chargement et initialisation de l'environnement

On commence par importer `labyrinthe.py`.

Puis on va créer une tâche labyrinthe à partir du contenu d'un fichier, ici `laby.1`. Pour cela, on tape :

```
laby = labyrinthe.Labyrinthe (filename = 'laby.1', x = 1, y = 1)
```

puis :

```
laby.reset ()
```

pour initialiser l'agent dans la case de coordonnées $(1, 1)$ qui sont spécifiées par les paramètres `x` et `y` lors de la création du labyrinthe.

2.2 Se déplacer

Ensuite, on peut se promener de case en case ; bien sûr, on doit rester dans les couloirs. On utilise la fonction `step()` en indiquant la direction en paramètre codée comme suit :

— 0 pour rester sur place,
— 1 pour aller une case à gauche,
— 2 pour aller une case à droite,
— 3 pour aller une case vers le haut,
— 4 pour aller une case vers le bas.

La fonction `step()` renvoie quatre valeurs :

— les coordonnées sous la forme d'un tableau de 2 éléments,
— le retour immédiat,
— le drapeau `done` qui indique si la case à atteindre a été atteinte,
— un dictionnaire qui contient un élément, le nombre de pas parcourus.

2.3 Autre fonction utile

La fonction `possible_actions ()` peut vous être utile : elle renvoie deux objets qui donnent la même information sous deux formes différentes :

— un tableau de 5 booléens indiquant pour chacune des actions si elle est possible dans la position courante. Une action est possible si elle n'est pas bloquée par un mur.
— Une liste contenant le numéro des actions possibles dans la position courante de l'agent.

3 À faire

1. **Implantation et mise au point du *Q-Learning***

   Implanter l'algorithme du *Q-Learning* vu en cours et décrit dans le poly pour résoudre ce problème de labyrinthe. On utilisera une stratégie de sélection gloutonne avec ϵ décroissant. On prendra $\gamma = 0{,}9$.

   On effectuera un certain nombre d'épisodes. Un épisode consiste à remettre à zéro de la position de l'agent (par appel à `reset()`) puis à effectuer le nombre d'itérations nécessaires pour atteindre la sortie.

   Réaliser un graphique affichant la courbe d'apprentissage, c'est-à-dire un graphique portant l'épisode en abcisses, le nombre d'itérations pour atteindre la sortie en ordonnées.

2. **Étude de la variabilité de la coure d'apprentissage du *Q-Learning***

   Réaliser un nombre raisonnable (en fonction de la durée d'exécution, 100 serait bien si c'est raisonnable) d'exécutions afin d'obtenir une estimation assez fiable du nombre d'épisodes à réaliser pour atteindre la sortie selon un chemin à peu près optimal, ainsi que sa variabilité.

   Réaliser un graphique de ces N courbes d'apprentissage.

3. **Influence de la valeur de γ**

   Est-ce que cela change quand on utilise d'autres valeurs de γ ? Par exemple, 0,5 ou 0,1.

4. Tester votre programme sur les 2 autres labyrinthes fournis. Réaliser les courbes d'apprentissage pour ces deux autres labyrinthes. Comparer.

5. **Expérience de changement de cibles**

   On s'intéresse maintenant au comportement du *Q-Learning* quand, ayant appris à atteindre rapidement sa cible, celle-ci change de position. On prendra le `laby.4` pour cette activité.

   (a) Entraîner l'agent de manière à ce qu'il atteigne la position cible de manière à peu près optimale.

   (b) Positionner la cible à la case de coordonnées $(14, 1)$. Pour cela, on utilise la fonction `set_sortie (x, y)`.

   (c) Remettre l'agent dans sa position initiale et le faire exécuter suffisamment d'épisodes pour qu'il atteingne la nouvelle position de manière à peu près optimale.

   (d) Positionner la cible à la case de coordonnées $(1, 2)$. Refaire la même chose.

   (e) Tracer la courbe d'apprentissage.

6. **Étude de SARSA**

   Refaire les mêmes activités avec l'algorithme SARSA décrit dans le poly (algorithme 5). Comparer les performaces du *Q-Learning* avec SARSA.