Learning Robots

alphai

COURS DECOUVERTE DE L'INTELLIGENCE ARTIFICIELLE

EVALUATION

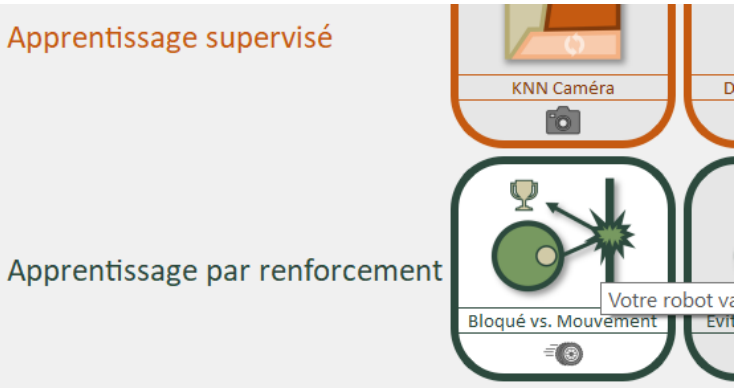
PROGRAMMATION DU Q-LEARNING

PROGRAMMATION DU Q-LEARNING

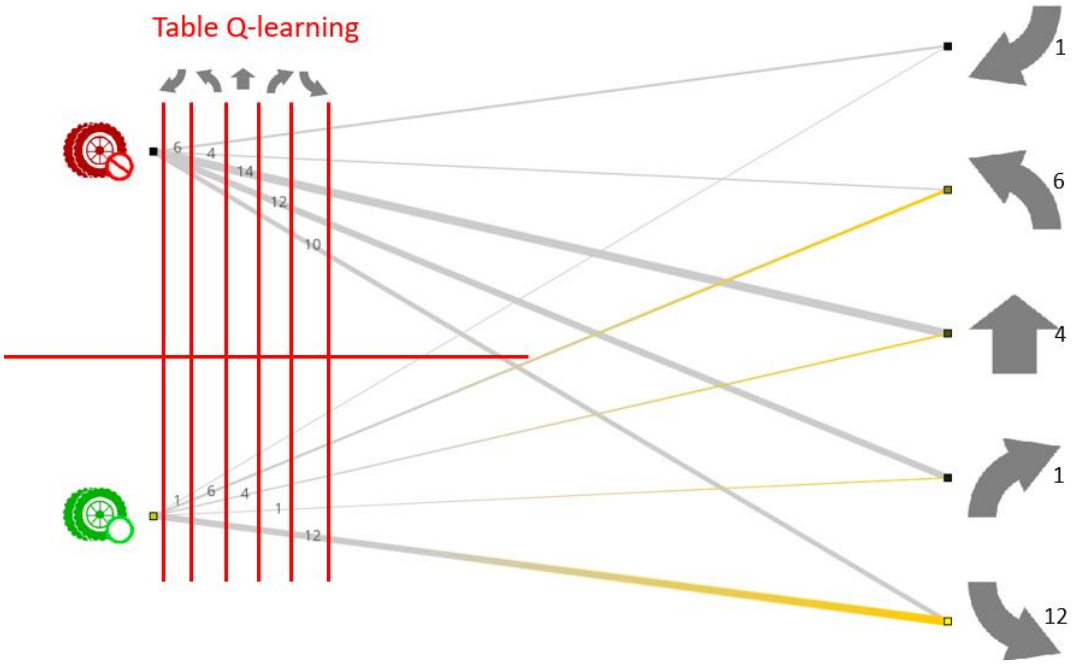
Au cours de cette évaluation vous allez créer un fichier Python : vous le déposerez sur la plateforme à la fin.

Introduction

Chargez la configuration de démonstration « Apprentissage par Renforcement – Bloqué vs. Mouvement ».



Nous avons vu lors de la séance précédente que ce qui est représenté à l'écran comme des connexions à l'intérieur d'un réseau de neurones est en réalité une table : la table des « Q-valeurs » de l'algorithme Q-learning.

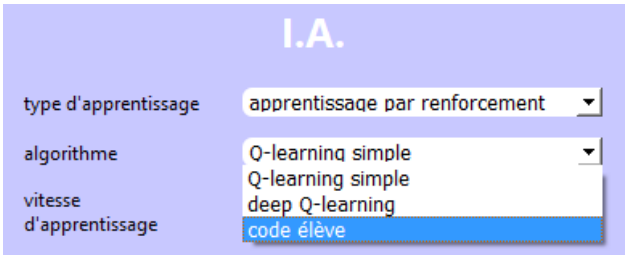


	Valeur de reculer à gauche	Valeur de tourner à gauche	Valeur d'aller tout droit	Valeur de tourner à droite	Valeur de reculer à droite
Quand le robot est bloqué	0.06	0.04	0.14	0.12	0.10
Quand le robot n'est pas bloqué	0.01	0.06	0.04	0.01	0.12

Initialement ces valeurs ne sont pas correctes. Au cours de l'apprentissage elles vont être modifiées, pour correspondre à des prédictions des récompenses qui pourront être reçues si on choisit telle ou telle action dans le cas « bloqué » ou « non bloqué ».

Démarrons la programmation de votre propre code du Q-learning !

1. Dans l'onglet IA, choisissez algorithme : code élève, puis « Nouveau », et choisissez le nom de fichier pour votre code (terminez bien par « .py » à la fin). Ouvrez-le dans votre éditeur favori.



Le fichier a trois fonctions :

- `init` est appelée quand on appuie sur « REINITIALISER L'IA »
  → C'est ici que nous allons initialiser la table
- `take_decision` est appelée à chaque fois que le programme a besoin de prendre une décision pour le robot
  → Nous allons commencer par cette fonction
- `learn` est appelée après qu'une décision a été prise
  → C'est ici que la table sera mise à jour
-

Appuyez sur « AUTONOME » 

: le robot se met à bouger (il tourne en arrière).

Prise de décision

La fonction `take_decision` est de prototype :

```
take_decison(state:int) -> action:int
```

Elle a pour entrée l'état `state` du robot (0 si le robot est bloqué, 1 si le robot n'est pas bloqué), et pour sortie le numéro de l'action que prend le robot.

2. Ajouter un print(state) pour afficher cet état dans la console

3. Pour l'instant, la fonction `decision` renvoie 0. A quelle action cela correspond-t-il ? Essayez de remplacer 0 par d'autres valeurs et notez à quelles actions cela correspond.

4. Remplacez le code de la fonction pour que le robot ait un fonctionnement cohérent : s'il est bloqué, faire un virage en arrière, sinon aller tout droit. Quel niveau le robot atteint-il ?

Pour permettre l'évaluation de votre travail, faites une copie en bas du fichier de la fonction take_decision et renommez cette copie « question4 ». (A la fin de la séance, vous rendrez le fichier Python). Par exemple :

```
def question4(state):
      if … :
            return …
      else:
            return …
```

A présent nous allons utiliser la table des valeurs pour prendre la décision !

5. Copiez le code suivant pour l'initialisation de cette table dans la fonction « init » :

```
qtable = None

def init(n_state, n_action):
    global qtable
    qtable = [
        [0.06, 0.04, 0.14, 0.12, 0.10],
        [0.01, 0.06, 0.04, 0.01, 0.12]
    ]
```

La variable `qtable` contient la table de Q-Learning présentée en introduction de cette partie. C'est donc un tableau 2D représenté en informatique par une liste de listes de nombres.

6. Modifiez la fonction `take_decision` pour que maintenant le robot choisisse :
   - s'il est bloqué l'action de valeur la plus élevée dans la première ligne de la table
   - s'il n'est pas bloqué l'action de valeur la plus élevée dans la deuxième ligne de la table

   Vérifier que le robot se comporte comme attendu : s'il est au milieu de l'arène il devrait toujours choisir de tourner en arrière droit.

Nous avons vu précédemment qu'il sera important que le robot effectue parfois des explorations.

7. Définissez en haut du fichier un paramètre EXPLORATION qui définit le niveau d'exploration, c'est-à-dire la probabilité (entre 0 et 1) de faire une exploration et choisir la prochaine action au hasard plutôt que de choisir l'action de valeur maximale selon la table. Par exemple :

```
EXPLORATION = 0.15
```

8. Améliorez la fonction `take_decision` pour que avec probabilité EXPLORATION l'action soit choisie au hasard entre les 5 actions possibles, et sinon on choisit la meilleur action selon la table (comme pour la question 10).

   Vous pourrez utiliser par exemple la librairie `random` : en haut du fichier

```
import random
```

   - la fonction `random.random()` renvoie un nombre réel entre 0 et 1
   - la fonction `random.randint(a, b)` renvoie un nombre entier entre a et b inclus

   Vérifiez que le robot choisit bien une action aléatoire de temps en temps.

Premier apprentissage

Introduisons à présent la fonction `learn` ! Elle est de signature

```
learn(old_state:int, action:int, reward:int, new_state:int) -> None
```

Son rôle est de modifier en place la table de Q-Learning en se basant sur :

- l'état précédent `old_state`
- l'action précédemment choisie `action`
- la récompense obtenue suite à cette action `reward`
- le nouvel état dans lequel le robot se trouve `new_state`.

9. Pour commencer, remplacez la valeur dans la table à la ligne `old_state` et à la colonne `action` par `reward`. Ajouter `print(qtable)` après la modification de la table pour pouvoir afficher les évolutions de la table dans la console.

Testez avec le robot : il devrait apprendre très vite à ne plus aller en arrière et à valoriser les actions qui vont en avant.

En revanche :

- les valeurs de ces actions risquent d'osciller
- il n'arrive pas à apprendre que la meilleure action à effectuer lorsqu'il est bloqué est de se retourner.

Nous allons corriger ces deux problèmes !!

Comme pour la question 4 précédemment, sauvez cette version de la fonction ***`learn`*** *en la copiant plus bas dans le fichier et en renommant la fonction* ***`question9`****.*

Amélioration des oscillations

Attention ! Nous allons écrire ci-dessous certaines formules mathématiques simples. Pour ces écritures, il est habituel d'utiliser d'autres symboles plus courts que les noms utilisés dans le code :

- l'état précédent, représenté par la variable `old_state` dans le code, sera noté s_t
- l'action précédemment choisie (`action` dans le code) sera notée a_t
- la récompense (`reward` dans le code) sera notée R_t
- le nouvel état (`new_state` dans le code) sera noté s_{t+1}
- une affectation, effectuée avec le symbole = dans le code, sera plutôt notée ←

Ainsi la valeur dans la table à la ligne `old_state` et à la colonne `action` est simplement notée $Q(s_t, a_t)$. t représente bien évidemment l'instant présent.

Commençons par régler le problème des oscillations.

Au lieu de remplacer la valeur $Q(s_t, a_t)$par la récompense R_t, nous allons plutôt « rapprocher légèrement » $Q(s_t, a_t)$ de R_t, avec une « vitesse d'apprentissage » notée α. La formule est :

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha R_t$$

10. Définissez en haut du fichier (juste en-dessous du paramètre EXPLORATION) le nouveau paramètre ALPHA = 0.2 et modifiez le code de la fonction `learn` pour mettre à jour la valeur $Q(s_t, a_t)$ comme indiqué ci-dessus.
    Relancez l'apprentissage, et vérifiez que les valeurs dans la table sont modifiées plus progressivement, ce qui résout le problème des oscillations.

Améliorer l'anticipation des récompenses futures

Enfin, pour que le robot apprenne à se retourner lorsqu'il est bloqué, il faut que la table des Q-valeurs ne soient pas des prédictions seulement de la récompense immédiate R_t, mais bien les récompenses suivantes également R_{t+1}, R_{t+2}, etc.

Plus précisément on introduit un nouveau paramètre γ légèrement inférieur à 1 et le but sera qu'au terme de l'apprentissage la valeur de Q représente la prédiction :

$$Q(s_t, a_t) \approx R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \cdots$$

On remarque que à l'instant t+1, $Q(s_t, a_t)$ devra lui représenter la prédiction :

$$Q(s_{t+1}, a_{t+1}) \approx R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$

Et donc que la fonction Q vérifie l'équation suivante (nommée « équation de Bellman » du nom de celui qui l'a prouvée formellement) :

$$Q(s_t, a_t) \approx R_t + \gamma Q(s_{t+1}, a_{t+1}),$$

Où a_{t+1} doit représenter la *meilleure* action pouvant être choisie à l'instant $t + 1$. C'est-à-dire telle que $Q(s_{t+1}, a_{t+1}) = \max_a Q(s_{t+1}, a)$.

Pour qu'à la fin de l'apprentissage la table des valeurs Q vérifie l'équation de Bellman, nous allons remplacer l'évolution ainsi :

$$target = R_t + \gamma \max_a Q(s_{t+1}, a)$$

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha\, target$$

11. Introduisez le nouveau paramètre GAMMA = 0.8 et mettez à jour la fonction learn selon la formule ci-dessus.
    Relancez l'apprentissage et vérifiez qu'à présent l'algorithme va bien apprendre

qu'il faut « se retourner lorsque le robot est bloqué », c'est-à-dire donner des valeurs plus élevées aux actions de retournement lorsque le robot est bloqué qu'aux autres actions.

Pour un code qui marche avec différentes configurations

Bravo vous avez fait l'essentiel !

A présent il ne reste plus qu'à modifier le code pour l'adapter à d'autres configurations où les nombres d'états possibles (`n_state`) et d'actions possibles (`n_action`) sont quelconques.

12. Modifiez la fonction `init` pour initialiser la table Q à la bonne taille en fonction de `n_state` et `n_action`, et la remplir avec des valeurs aléatoires.
    Vérifiez le bon fonctionnement en ajoutant le capteur ultra-son (mode absence/présence d'obstacle) et utilisant de nouvelles actions.

A la fin de la séance, déposez votre fichier Python sur la plateforme du cours.