Learning Robots

alphai

L'EDUCATION À L'INTELLIGENCE ARTIFICIELLE

COURS DECOUVERTE DE L'IA

EVALUATION « TOUR DE PISTE »

Evaluation « Tour de piste »

Cours Découverte de l'IA – 10 novembre 2022

Créez un dossier pour l'évaluation avec votre nom : par exemple « IA_evaluation1_ThomasDeneux ». A la fin de l'évaluation vous le **compresserez** et me **l'enverrez** au travers de la boîte « Rendu Evaluation » sur la page du cours.

Partie 1 – Avec le capteur ultra-son

Nous allons faire se « balader » le robot simulé dans l'arène grâce à ses capteur de blocage et de distance ultra-son, avec 3 méthodes différentes.

Question 1. Apprentissage supervisé KNN avec Ultra-son

Démarrez l'ordinateur sous linux. Démarrez le programme alphai (dans un terminal, ~tdeneux/alphai). Un nouvel écran bleu apparaît, sélectionnez « Simulation ».

Nous allons configurer un apprentissage supervisé KNN avec capteur de blocage et ultra-son :

1. Ajoutez le capteur ultra-son : 

. Ce capteur renvoie la distance au premier obstacle.
2. Gardez le capteur de vitesse (qui détecte les blocages), mais pour qu'il ne « fabrique » que 1 entrée au lieu de 2, dans l'onglet IA décochez « 2 neurones par variable binaire ».
3. Sélectionnez l'algorithme d'apprentissage supervisé « K plus proches voisins ».

Créez dans le dossier de l'évaluation un sous-dossier « question1 ».

Sauvez la configuration à l'intérieur de ce dossier (menu : Paramètres > Sauver les paramètres ; vous pouvez garder le nom par défaut « parameters.json »).

A présent, entraînez le robot simulé à se « balader » dans le circuit ; il doit :

- Aller tout droit lorsqu'il y a suffisamment d'espace
- Tourner avant les obstacles
- Se dégager s'il se cogne malgré tout dans un obstacle

Il n'y a pas besoin que le robot fasse des tours de piste complets.

Vérifiez que le robot a bien appris en activant l'autonomie pendant au moins deux minutes.

Sauvegardez le résultat de l'apprentissage (menu : IA > Sauver l'état du réseau), toujours dans le dossier « question1 ».

Question 2. Même apprentissage avec un réseau de neurones.

Vous allez répéter le même apprentissage, mais cette fois en utilisant un réseau de neurone.

Pour voir à la fois le réseau de neurones et l'espace des états, dans l'onglet Visualisation sélectionnez tout en haut l'affichage « espace d'états et réseau ».

Pour l'entraînement, pas besoin de recréer un dataset ! chargez simplement la mémoire d'expérience de l'apprentissage précédent (menu : IA > Charger la mémoire d'expérience).

Attendez que l'IA apprenne correctement ; pour un apprentissage plus efficace augmentez le nombre de neurones / de couches de neurones !

Une fois l'apprentissage terminé, vérifiez que le robot se comporte bien en activant l'autonomie, puis sauvez vos résultats : **créez un nouveau sous-dossier** « question2 » dans le dossier principal ; **sauvez-y la configuration** ; et **sauvez également l'état du réseau** de neurones (répondre Oui à la question « Sauver aussi la mémoire d'expérience ? »).

Question 3. Comportement équivalent en programmation Python déterministe.

A présent, vous allez programmer vous-mêmes le même comportement avec Python !

Choisissez l'algorithme « code élève » comme vous l'avez fait lors du TP KNN. Créez un **nouveau fichier .py**, à l'intérieur d'un **nouveau sous-dossier « question3 »**. Sauvez également dans le même sous-dossier la **configuration** (menu : Paramètres > Sauver les paramètres).

Rappel :

- take_decision(sensors) prend en argument la liste des valeurs des capteurs ; ici sensors[0] est la valeur du capteur vitesse (0 ou 1) et sensors[1] est la valeur du capteur ultra-son (entre 0 et environ 3.5 ; utiliser des *print* pour afficher les valeurs).
- La fonction doit renvoyer un entier représentant le numéro d'action choisie (ici entre 0 et 4).
- Pour que le robot démarre, activer l'autonomie !

Vous n'avez pas à coder d'algorithme d'apprentissage ! Mais seulement la fonction take_decision, pour prendre les bonnes décisions en fonction de l'état des capteurs, en utilisant des blocs de contrôle *if… elif… else*. Le robot doit :

- Aller tout droit lorsqu'il y a suffisamment d'espace
- Tourner avant les obstacles
- Se dégager s'il se cogne malgré tout dans un obstacle -> Pour ne pas faire un code trop complexe, ne vous souciez pas des cas où le robot se bloque sur la séparation noire au milieu de la piste ; si cela arrive, appuyez simplement sur le bouton « Réinitialiser l'IA ».

Une fois que votre robot avance comme il faut, passez à la suite !

Partie 2 – Avec la camera

Vous pouvez faire cette partie sans avoir fait les questions précédentes. Le but est que le robot apprenne tout seul à faire des tours de piste, en apprentissage par renforcement, grâce à sa caméra.

Question 4. Apprentissage supervise avec caméra.

Chargez la configuration de démonstration « Course de robot ». Réduisez la résolution de la caméra à 16x12. Puis entraînez le robot à faire le tour de l'arène.

Créez un **nouveau dossier** « question4 » pour y sauver la **configuration** et **l'état du réseau** (répondre Oui à « Sauver également la mémoire d'expérience ? »).

Question 5. Apprentissage par renforcement avec caméra.

A présent, passez en apprentissage par renforcement (algorithme « deep Q-learning »). Démarrez l'apprentissage : le robot est récompensé lorsqu'il avance, et puni (récompense négative) lorsqu'il est bloqué, il apprend relativement rapidement à se « balader » dans l'arène, mais pas forcément à faire des tours de circuit dans un sens précis.

Vous pouvez accélérer l'apprentissage en appuyant sur le bouton « simulation accélérée », ainsi qu'en obligeant le robot à essayer certaines actions en le contrôlant avec les flèches. N'attendez pas d'avoir un apprentissage parfait mais passez plutôt à la question suivante.

Créez un **nouveau dossier** « question5 » pour y sauver la **configuration** et **l'état du réseau** à l'issue de l'apprentissage.

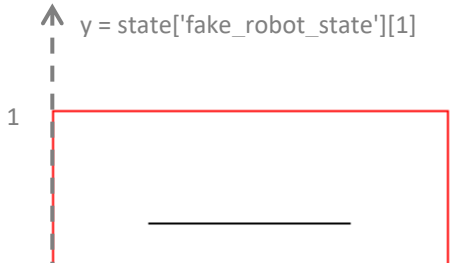
Question 6. Programmation de la récompense.

Nous voudrions que le robot tourne à l'intérieur de l'arène toujours dans le même sens, par exemple celui des aiguilles d'une montre. Par ailleurs nous voudrions qu'il tourne le plus vite possible.

Pour cela, vous allez programmer vous-même une nouvelle récompense, et récompenser le robot lorsqu'il tourne dans le bon sens !

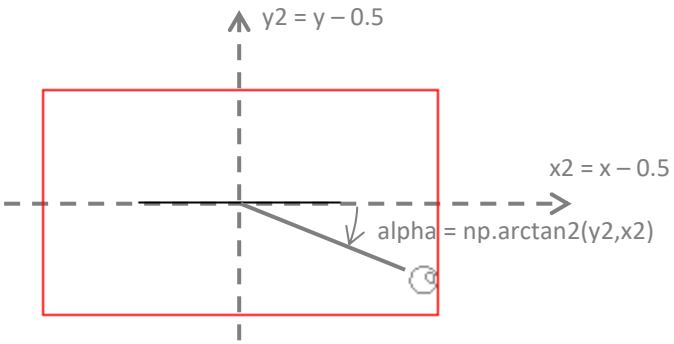
- Dans l'onglet Récompense, sélectionnez le mode de récompense « récompense élève ». Choisissez « Nouveau » pour créer un nouveau document Python. **Créez-le dans un nouveau dossier** « question6 ». Sauvez aussi dès à présent **la configuration** dans ce dossier.

- Vous allez programmer la fonction `reward(state)`. Cette fonction prend en entrée un dictionnaire avec certaines informations sur le robot : ajoutez un `print(state)` dans la fonction pour les afficher dans la console (démarrer le robot avec le bouton « autonome » pour que la fonction soit appelée). Vous allez utiliser la position du robot : définissez et affichez avec print `x = state['fake_robot_state'][0]` et `y = state['fake_robot_state'][1]`, compris entre 0 et 1.

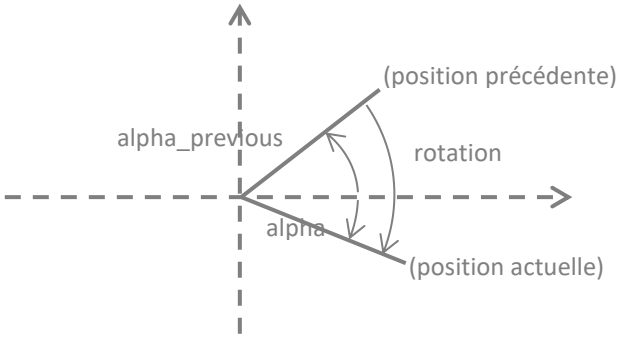


- La fonction `reward` renvoie la récompense. Si vous renvoyez `y` par exemple et démarrez un nouvel apprentissage (appuyer sur « Réinitialiser l'IA »), le robot va apprendre à se mettre tout en haut de l'arène ! (Vous pouvez essayer, mais ne perdez pas trop de temps, passez à la suite).

Pour récompenser le robot lorsqu'il tourne dans le sens des aiguilles d'une montre, nous allons calculer sa rotation autour du centre de l'arène. Voici ci-dessous les formules mathématiques pour obtenir la coordonnée angulaire alpha du robot (c'est-à-dire l'angle entre la direction horizontale et la direction du centre de l'arène vers le robot).



Pour mesurer la rotation, il faut comparer la valeur actuelle de l'angle `alpha` avec la valeur précédente (appellons-la `alpha_previous`).



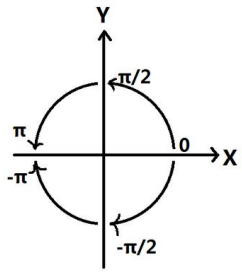
Pour pouvoir faire le calcul il faut avoir accès à la valeur précédente de l'angle `alpha_previous`. Vous la sauverez dans une variable globale : votre code ressemblera à ceci (il ne reste plus qu'à compléter les parties manquantes…).

```
import numpy as np
```

```
alpha_previous = 0

def reward(state):
    global alpha_previous
    ...
    alpha = ...
    rotation = ...
    alpha_previous = alpha
    return rotation
```

Attention !, pour le calcul de la rotation il faudra prendre en compte que la fonction `np.arctan2` renvoie une valeur entre -π et π, et donc il faudra gérer un cas particulier lorsque le robot franchit la ligne des x négatifs.



atan2

- Complétez la fonction reward, puis testez-la en lançant l'apprentissage (après chaque changement du code, cliquez « Réinitialiser l'IA »).

Sauvez l'état du réseau de neurones une fois qu'il a appris à faire le tour de l'arène du mieux possible !