T-AIA 902

Taxi Driver V3

ROUSSEL Rémi, BRILLARD Quentin, HERTZ Artur, LUCAS Michaël, TROUSSET Alexis

Sommaire

1 Qu'est-ce que TAXI V3

Taxi V3 est un environnement de Gym qui simule un Taxi au sein d'une ville. Au travers de cette environnement, nous devons contrôler ce Taxi afin de "résoudre" le jeu. Nous pouvons visualiser la ville avec la grille ci-dessous.

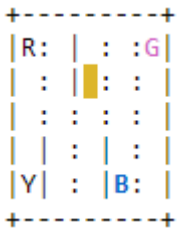

Figure 1 : La grille de Taxi V3

1.1 Explication de la grille

La grille ci-dessus représente donc une configuration de la ville dans laquelle nous sommes censés évoluer avec le taxi. Sur cette capture, nous pouvons discerner quatre éléments différents :

- Les points d'actions (R,G,Y,B) points ou nous pouvons récupérer ou déposer un client
- Le taxi, représenté par la barre jaune (se colorise en vert si un client est abord du taxi)
- Des murs représentés par le symbole '|', le taxi ne peut pas passer au travers de ces derniers
- Des séparations de voies, représentés par le symbole ':'

1.2 Liste des actions

Afin de se déplacer dans cette ville et réaliser les actions de récupération / dépôt de client nous avons différentes actions disponibles. En effet pour résoudre ce jeu, nous pouvons opérer six actions distinctes :

- Déplacer le taxi vers le Sud
- Déplacer le taxi vers le Nord
- Déplacer le taxi vers l'Est
- Déplacer le taxi vers l'Ouest
- Récupérer un client
- Déposer un client

Ces actions, nous pouvons essayer de les réaliser à n'importe quel endroit de ville. Cependant en fonction de l'endroit (ou état) où nous sommes ces actions vont avoir des résultats différents.

1.3 Résultats liées aux actions

Les différents résultats sont au nombre de trois et dépendent directement de l'action réalisé en fonction de l'état. Nous pouvons donc déduire que le résultat peut se traduire via la fonction suivante :

Résultat = f(action, state)

Figure 2 : Équation d'obtenions d'un résultat

Le résultat peut-être de trois natures différentes :

- Bon, équivaut à +20 points, c'est pour chaque récupération / dépôt de clients correct
- Mauvais, équivaut à -10 points, c'est pour chaque fois qu'on va essayer de récupérer / déposer un client alors que nous ne sommes pas à l'endroit prévu pour, ou tourner dans un mur
- Neutre, équivaut à -1 point, pour chaque action n'étant ni mauvaise, ni bonne (un déplacement n'étant pas dans un mur)

2 Résolution de TAXI V-3

Pour relever le défi proposé par cet environnement, nous avons prévus de réaliser trois algorithmes différents basés sur des principes d'intelligence artificielles différentes.

2.1 Le Temporal Difference Learning (TD)

Le Temporal Difference (TD) learning est une des classes d'algorithmes d'apprentissage par renforcement. Sa particularité est qu'elle ne possède pas de modèle. C'est à dire, qu'elle n'utilise pas de fonction pour attribuer des récompenses comme le Processus de Décision de Markov (MDP) par exemple. Elle est utile dans des cas ou la récompense est simple à déterminer et est fixe. Le TD Learning tant à utiliser une approche utilisant les probabilités.

2.1.1 Le Q-Learning

Le Q-Learning, est une technique ou l'agent (le programme) va exécuter une action "a" en fonction de l'état "s" et de la fonction "Q". Il reçoit donc un nouvel état "s" ainsi qu'une récompenses "r". C'est alors qu'il met à jour la fonction "Q" en prenant en compte la récompense reçue ainsi que le nouvel état.

A présent après avoir expliqué le principe du QLearning, nous allons voir comment nous avons mis en place cet algorithme pour résoudre Taxi-V3. Pour commencer il fallait stocker les différents résultats obtenus en fonction des action réalisées pour un état donnée. Pour stocker ces données nous avons mis en place un tableau permettant de relater le résultat obtenu en fonction de l'action faite à un état donné.

2.1.1.1 Le Q-Table

Ce tableau porte pour nom 'Q-Table', en référence au nom de l'algorithme. En axis de ce tableau nous avons les différentes actions possibles et en ordonnées les états, c'est à dire où se situe le taxi dans la ville. Au début de l'algorithme, nous initialisons le tableau qu'avec des '0' et durant l'exécution de l'algorithme, ce tableau se mettra à jour.

Action/ State	Nord	Sud	Est	Ouest	Prendre	Dépôt
Pos. 0	0	0	0	0	0	0
…	…	…	…	…	….	…
Pos. 500	0	0	0	0	0	0

Figure 3 : Q-Table initialisé

2.1.1.2 Définition des paramètres d'exécution

Pour l'exécution de cet algorithme, nous devons définir quelques paramètres, comme le nombre d'épisodes d'entrainement (training_episodes), les épisodes d'actions (display_episodes), le taux d'apprentissage (alpha), le taux de décompte (gamma), ainsi qu'un facteur permettant de pondérer la chance de choisir une action aléatoire ou une action permettant de maximiser la récompense (epsilon).

2.1.1.3 Différence entre exploration et exploitation

D'après les principes de l'apprentissage par renforcement, nous pouvons faire deux types d'actions. L'exploration, le principe de faire une action aléatoire afin de remplir les valeurs du QTable. Le second type d'action c'est l'exploitation. Comme son nom l'indique, nous allons exploiter les données du tableau afin d'exécuter la meilleure action en fonction de l'état dans lequel nous nous trouvons.

Il est donc important dans un premier temps de faire un maximum d'exploration afin de remplir le QTable pour être capable par la suite de faire une exploitation optimale et résoudre rapidement le 'jeu'. Nous pouvons donc résumer cette situation par le schéma suivant.

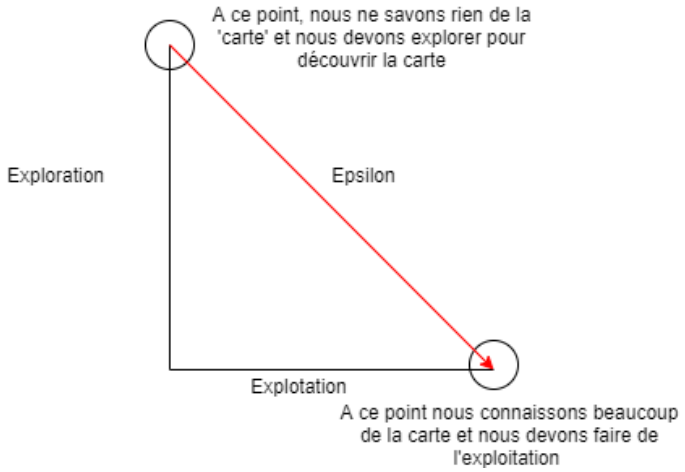

Figure 4 Enchaînement entre exploration et exploitation

D'après le schéma ci-dessus nous nous comprenons que le facteur Epsilon joue son rôle de bascule entre de l'exploitation et de l'exploration. En effet, au fil des actions, nous allons faire glisser cet indicateur vers le point pour maximiser les chances de faire de l'exploitation et non de l'exploration.

2.1.1.4 Phase d'entrainement

Durant cette phase, nous allons faire un maximum d'exploration afin de remplir au maximum le QTable et d'explorer tous les cas de figure possible. C'est pour cela qu'il faut définir un grand nombre d'épisodes d'entrainement (training_episodes).

L'environnent Taxi-V3 de gym nous retournes quatre informations après chaque actions (déplacement ou autre) :

- Le nouvel état (next_state)
- La récompenses (20, -1, -10) en fonction du type de récompenses (bon, neutre, mauvais) (reward)
- Une information si l'épisode est terminé (done)
- Ainsi qu'un log sur l'action de l'ont vient d'exécuter (info)

Après avoir exécuté cette action nous devons donc lui définir une valeur, pour cela, nous devons récupérer l'ancienne valeur, c'est à dire la valeur du q-table à l'état ou nous étions ainsi que l'action que nous venions de faire.

```
old_value = q_table[state,action]
```

Figure 5 : Calcul de l'ancienne valeur

Après avoir récupéré cette ancienne valeur nous devons récupérer la meilleure action à faire dans l'état ou nous somme. C'est à dire lire la ligne du tableau et récupérer l'action à la plus grande valeur de cette ligne. Ce qui permet de déduire en soit la meilleure action à faire dans cet état-là.

```
next_max = numpy.argmax(q_table[nex_state])
```

Figure 6 : Récupération de la meilleure action du nouvel état

Suite à cela, nous pouvons mettre à jour la case du QTable pour l'état auquel nous étions ainsi que l'action que nous venons de faire. Pour cela, nous allons utiliser l'équation de Bellman. Ce qui nous donne l'équation suivante :

```
new_value = (1 – alpha) * old_value + alpha * (reward + gamma * next_max)
```

Figure 7 : Utilisation de la formule de Bellman pour calculer la valeur de de l'action réalisée

2.1.1.5 Phase d'exécution

Après l'entrainement vient la compétition. Pour cette dernière phase dans l'algorithme de QLearning, nous allons mettre à l'épreuve les informations collectées durant cet entrainement. Pour cela nous allons évaluer sur un nombre assez faible d'épisodes (display_episodes), le nombre moyen d'étape pour terminer un épisode ainsi que le nombre moyen de pénalités reçues par épisodes.

Pour cette phase, nous ne faisons que de l'exploitation en cherchant à chaque état quel est la meilleure action à faire. C'est pour cela que dans cette étape, nous ne mettons pas à jour le QTable.

2.1.1.6 Résultat du QLearning

Après 20000 épisodes d'entrainement et 10 épisodes "d'évaluations". Nous obtenons les résultats suivants :

- Nombre moyen d'étapes par épisodes : 13.7
- Nombre moyen de pénalités par épisodes : 0

Nous avons décidé d'utiliser 20000 épisodes d'entraiment car en dessous de ce chiffre, le taux de réussite de l'algorithme n'était pas optimal, il était en dessous des 100%.

2.2. Monte Carlo learning

Contrairement au Temporal Difference learning les méthodes de Monte-Carlo utilisent une approche aléatoire pour déterminer ensuite à l'aide d'une formule la "valeur" d'un état et ainsi d'y attribuer une récompense. Contrairement au TD Learning l'attribution des récompenses se déroulent donc au moment où l'issue finale est connue d'où l'approche aléatoire.

Initialement conçue pour le Casino, cette approche est très courante et est très utile pour résoudre des problèmes où l'on ne peut estimer la valeur d'un état, un exemple courant est celui du BackJack où la personne doit piocher jusqu'à s'approcher le plus possible de 21. Ayant une dimension probabilistique, l'estimation de la récompense à attribuer à une action ne peux être réalisé qu'une fois que le resultat final est annoncé.