

# IA - Taxi driver

## Reinforcement learning

1.	Présentation du projet	3
2.	Algorithmes	3
2.1.	Brute force	3
2.2.	Q-learning	4
2.3.	Deep Q-learning	5
2.4.	Algorithme de Monte-Carlo	5
3.	Choix et justifications	6

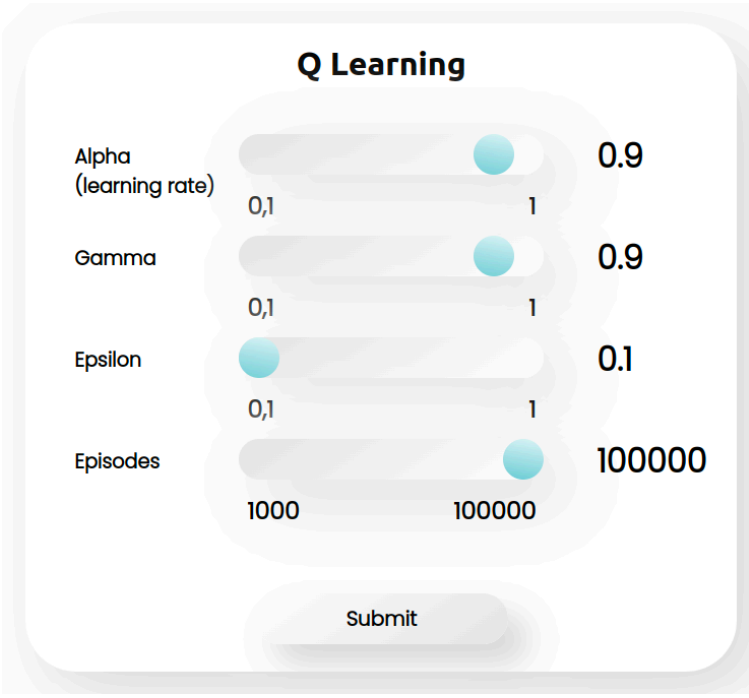
## 1. Présentation du projet

Le projet Taxi Driver consiste à développer un algorithme permettant de résoudre un problème. La mise en situation est un taxi qui doit prendre un passager présent à un endroit aléatoire de la carte et le déposer à l'un des quatre endroits possibles. L'algorithme doit trouver la façon la plus rapide de déposer le passager au bon endroit.

Deux modes sont disponibles :

- Un mode utilisateur qui permet de régler les paramètres de l'algorithme
- Un mode limité dans le temps où tous les paramètres sont optimisés afin de réduire le nombre d'étapes pour résoudre le problème dans un temps donné

### Exemple du mode utilisateur



The image shows a user interface for configuring Q Learning parameters. It features four sliders, each with a label, a range, a current value indicator (a blue dot), and a numerical value. The parameters are Alpha (learning rate), Gamma, Epsilon, and Episodes. A 'Submit' button is located at the bottom.

Parameter	Range	Current Value
Alpha (learning rate)	0,1 to 1	0.9
Gamma	0,1 to 1	0.9
Epsilon	0,1 to 1	0.1
Episodes	1000 to 100000	100000

## 2. Algorithmes

### 2.1. Brute force

La recherche par force brute est une méthode algorithmique qui consiste principalement à essayer toutes les solutions possibles. Par exemple pour trouver

le maximum d'un certain ensemble de valeurs, on consulte toutes les valeurs possibles jusqu'à trouver la solution, le tout sans logique ou réflexion.

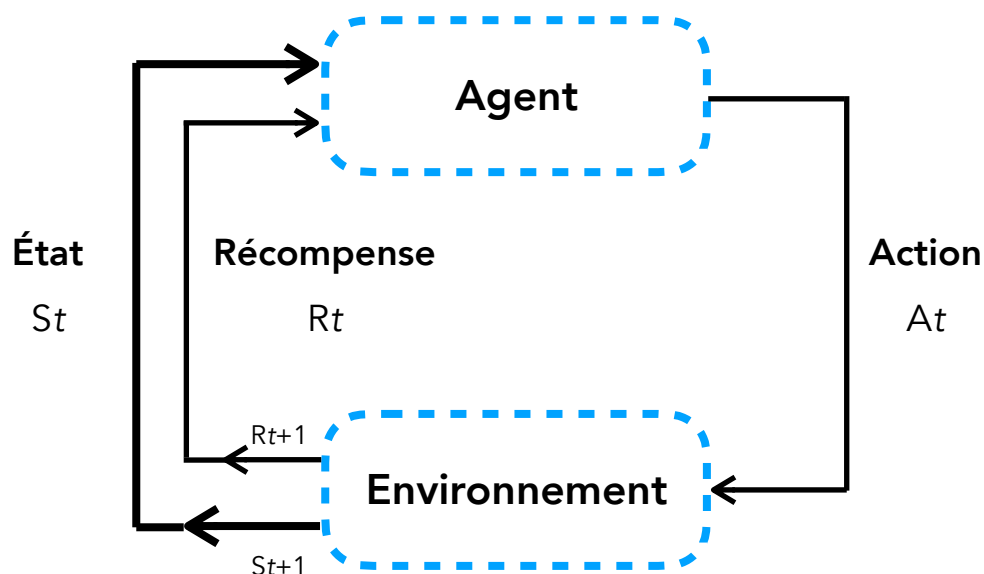
Dans notre cas, le taxi va se déplacer aléatoirement sur la carte. Avant chaque déplacement nous allons vérifier si nous avons la possibilité de prendre ou de déposer un passager.

## 2.2. Q-learning

Le Q-learning est un type d'algorithme basé sur un système de récompense. C'est une technique d'apprentissage par renforcement qui ne nécessite aucun modèle initial de l'environnement. C'est un algorithme "off-policy", autrement dit c'est un algorithme qui ne nécessite pas de contrôle lors de son processus d'apprentissage.

Nous attribuons un score de base de 50 à l'algorithme puis, à chaque mouvement, 1 point est retiré. Lorsque le tableau est fini nous ajoutons 100 à ce score. Ce score final sera ajouté dans chaque case de la grille sur laquelle nous nous serons déplacés lors de la réalisation de l'exercice. Cette manœuvre va ensuite être répétée le plus de fois possible afin que chaque case possède un jeu de données pertinent. Cette phase est appelée de "training".

### Schéma de la phase de learning



Pour finir une fois l'algorithme entraîné il peut résoudre les problèmes qui lui sont donnés. Lors de l'exécution il va se déplacer à chaque mouvement vers les cases avec les plus gros scores car ce sont celles qui auront nécessité le moins de déplacement lors de l'entraînement pour résoudre le problème.

Le problème rencontré avec le Q-learning est que si l'algorithme est trop entraîné, le jeu de données peut être trop important. Ce qui par la suite cause des baisses de vitesse d'exécution et donc une baisse de performances.

### 2.3. Deep Q-learning

Le deep Q-learning est une version plus complexe du Q-learning que nous avons présenté plus haut. C'est un algorithme d'apprentissage par renforcement profond. L'apprentissage par renforcement intègre l'apprentissage profond dans la résolution, permettant aux agents de prendre des décisions à partir de données d'entrée réorganisées sans intervention manuelle. Ces algorithmes sont capables de prendre en compte de très grandes quantités de données et de décider des actions à effectuer pour optimiser un objectif.

Pour l'entraînement de notre algorithme, une variable appelée *Epsilon-Greedy* est utilisée. Cette variable introduit la possibilité d'effectuer un coup aléatoire. Au début de l'entraînement chaque coup est effectué de manière aléatoire car *Epsilon-Greedy* vaut 1. Au fur et à mesure que l'entraînement avance nous allons lisser cette variable afin qu'elle s'approche de 0. Cela réduira donc la probabilité d'avoir un coup aléatoire.

Le réseau neuronal est là pour apprendre et évoluer à chaque exécution de l'algorithme, à chaque fois qu'il fini l'algorithme il évalue ses actions et leur donne un poids, ce poids varie en fonction du résultat. Grâce à cela, une fois entraîné le réseau neuronal sais quelle action effectuer suivant un contexte précis, car chaque action possède un poids qui lui est propre.

Dans notre cas nous utilisons le deep Q-learning fonctionne avec un système de prédiction. Nous créons un réseau de neurones qui va analyser la valeur de chaque coup avant de l'effectuer, afin de sélectionner le coup le plus rentable et le plus pertinent à la résolution du problème du taxi.

### 2.4. Algorithme de Monte-Carlo

Un algorithme de Monte-Carlo est un algorithme qui possède deux caractéristiques principales :

- Il est randomisé, c'est-à-dire qu'il utilise un aléa au cours de son calcul.
- Son temps d'exécution est fixe. Sa nature aléatoire se manifeste dans son résultat qui peut être incorrect avec une certaine probabilité (généralement minime), mais qui peut-être quantifiée rigoureusement.

L'avantage est que le temps de résolution du problème est défini avant le lancement de l'algorithme, on sait donc déjà à quel point il sera rapide. En revanche, son résultat a des chances d'être faux ce qui est un désavantage assez important.

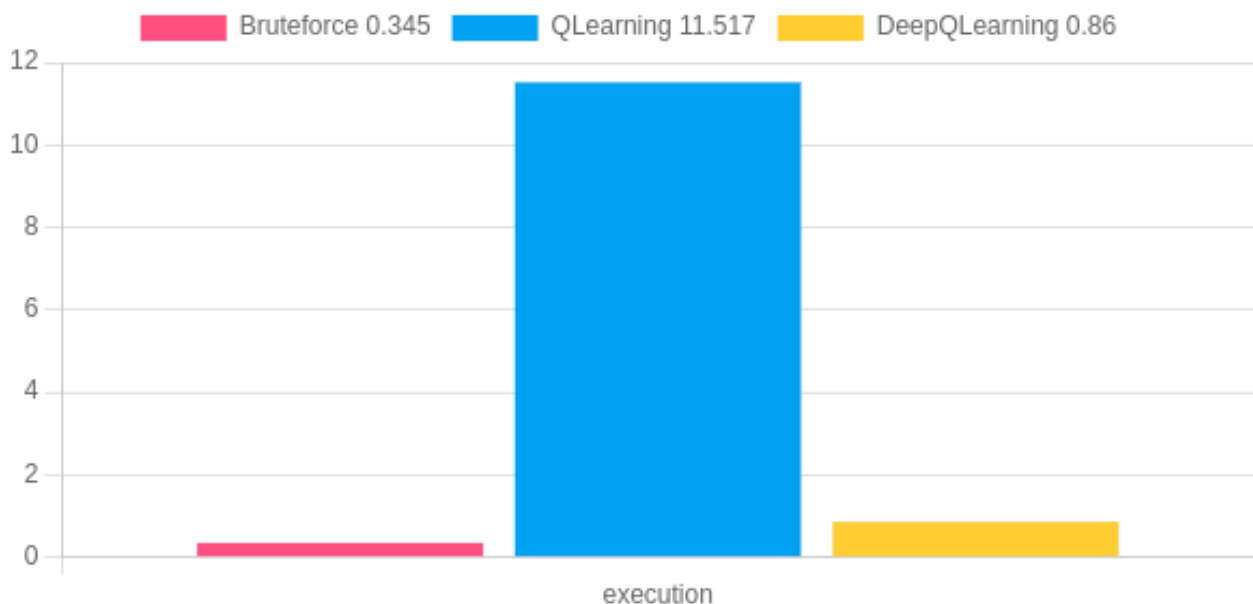
### 3. Choix et justifications

Durant le développement du projet nous avons choisi de développer 3 algorithmes afin de les comparer et décider lequel serait le plus performant et pertinent pour résoudre le problème qui nous est posé : le brute force, le Q-learning et le deep Q-learning. Afin de comparer les performances et les résultats de chaque algorithme nous avons récupéré les données d'exécution et de résolution puis nous les avons traités et affichés sous forme de graphiques sur une page web.

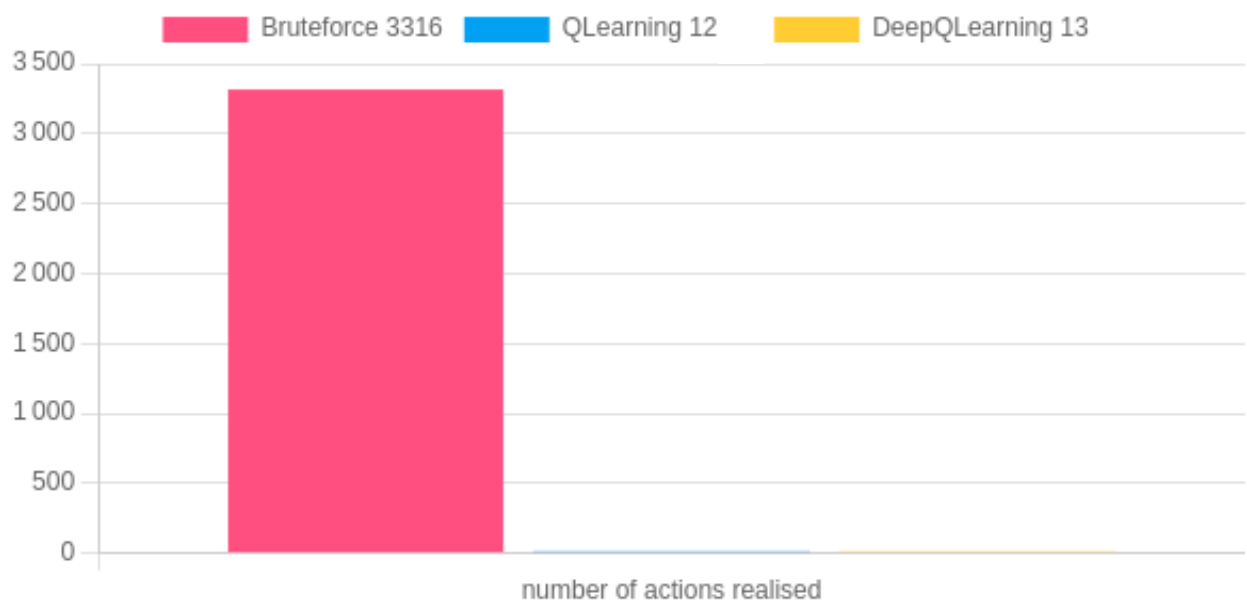
Trois indicateurs principaux ont été définis pour déterminer quel algorithme était le mieux adapté à notre situation :

- Le temps d'exécution
- Le nombre d'erreurs commises (non respect des règles du problème / erreur de résolution)
- Nombre d'actions réalisées avant de résoudre le problème

L'algorithme le plus performant dans les trois domaines sera celui qui sera retenu pour la résolution de notre problème.

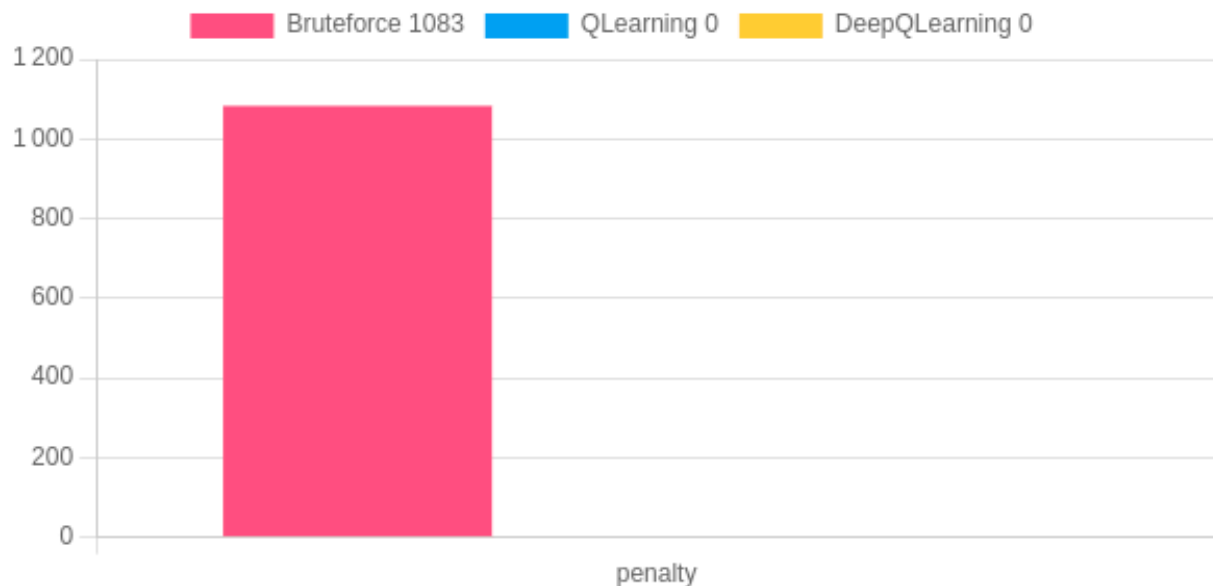


Concernant le temps d'exécution on remarque que le brute force est le plus rapide mais que le deep Q-learning s'en rapproche. En revanche, comme précisé dans la description des algorithmes, le Q-learning est beaucoup plus long à s'exécuter à cause d'une quantité de données trop importante qui ralentit le processus.



Au niveau du nombre d'actions réalisées on peut voir sans surprises que le brute force fait beaucoup plus d'actions que les algorithmes ayant appris au préalable quelles étaient les meilleures solutions. Malgré sa vitesse d'exécution le

brute force n'est donc pas optimisé en terme de nombre de déplacement effectués pour résoudre le problème. Le Q-learning et le deep Q-learning sont quasiment au même niveau de performances et ne peuvent donc pas être départagés sur ce simple critère.



Et enfin, au niveau des erreurs effectuées, le Q-learning et le deep Q-learning sont irréprochables. Contrairement au brute force qui produit bien trop d'erreurs et n'est donc pas fiable pour la résolution de notre projet.

Pour conclure grâce à ces graphiques nous pouvons en déduire que grâce à sa vitesse d'exécution très rapide, l'absence d'erreurs lors de la résolution et du peu de coups nécessaire pour résoudre le problème, le deep Q-learning est le meilleur algorithme à utiliser dans le cas du taxi driver.