T-AI-902 TAXI

Romain PIOT

Jatin BHATIA

Bastien BERGUER

Problématique

Le projet Taxi Driver a pour objectif de s'initier au méthode de renforcement learning. Le projet consiste à créer une intelligence artificielle capable de jouer au jeu Taxi-v3 et de gagner le plus efficacement possible.

Différentes approches étaient recommandées, le but étant de comparer les performances des IA issues de ces différentes approches.

Méthode de travail

1. Discussion au début du projet de différentes approches/technologies pour réaliser le projet
2. Veille technique et étude de la faisabilité des approches, chaque personne du groupe travail sur une approche
3. Utilisation d'un trello pour lister les différentes tâches et les assigner
4. Une réunion hebdomadaire sur Teams pour traiter des problèmes techniques de chacun et l'avancement du projet
5. L'utilisation de branche git pour les différente parties

Approches

I. Q learning

A. Définition

Q-learning est un algorithme d'apprentissage par renforcement, via un agent qui apprend de lui-même. En fonction des choix effectués, il va apprendre, et mettre a jours sa table de connaissance (la q_table), en y mettant une valeur de récompense plus ou moins élevée en fonction du choix effectué pour une action donnée.

C'est un algorithme sans policy (sans politique). C'est à dire que nous ne programmons aucune règle ou principe afin de faire des choix, il les fait lui-même, en respectant les principe même du q-learning.. C'est à dire que sont principe de réalisation s'appuie sur le fait de récupérer la meilleure récompense possible, et donc d'effectuer les choix ayant le plus de chance d'aboutir à la résolution du problème rencontré.

Le Q de Q-learning est la valeur qualitative avec laquelle le modèle va trouver la prochaine action améliorant la qualité.

B. Réalisation

Afin de permettre à l'algorithme d'apprendre et de jouer de façons différentes, nous avons défini des paramètres modifiables pour chaque algorithme, et c'est le cas pour notre q-learning. De ce fait, il est possible de modifier l'epsilon, l'epsilon min et max, le learning rate, la reward discount rate, le decay rate, le nombre max de step réalisé par jeu, et le nombre d'épisodes souhaités.

Lorsque l'agent va parcourir les différentes possibilités du jeu, il va mettre à jour sa q_table, et continuer à jouer et mettre à jour les données qu'il aura récolté.

Afin de permettre à l'agent d'apprendre mais aussi d'utiliser ses connaissances récoltées au fil des parties, nous utilisons la méthode d'epsilon-greedy.

Cette méthode permet de mettre à jour l'agent au fil du jeu, afin qu'il ai tendance quasiment systématiquement faire de nouveau choix encore jamais effectué en début de parti, puis au fil du temps, utiliser de plus en plus ses connaissances, afin de s'appuyer sur les récompenses qu'il aura eu par le passé afin de faire les meilleurs choix possible.

Une fois le jeu terminé, lorsqu'il à parcouru l'ensemble des épisodes, l'algorithme retourne sa moyenne de récompense, sa moyenne de step nécessaire à la résolution du jeu, ainsi que sa durée totale d'exécution.

II. Deep Q Learning - Deep RL

A. Définition

Le concept du deep RL est de combiner l'apprentissage par renforcement expliqué plus haut et des méthodes de deep learning.

B. Réalisation

Pour réaliser cette partie nous avons créé un modèle de deep learning assez simple car le jeu ne possède pas un grand nombre d'action et de données possible.

Un fonctionnalité de sauvegarde permet de conserver les modèles entraînés pour les réutiliser et les comparer.

Pour évaluer l'évolution des performances des modèles nous avons choisi de sauvegarder des modèles ayant été entrainé entre 100 000 partie et 1 000 000 (tous les 100 000).

Nous avons automatisé l'entraînement ainsi que la sauvegarde des modèles. Pour optimiser le temps de l'entraînement nous avons eu recours à l'utilisation d'un gpu. L'entraînement de 1000 000 parties a malgré tout duré plus de 40 minutes.

En terme de paramètre pour l'entraînement après plusieurs essais nous avons configuré le learning rate a 1e-3. Nous avons utilisé le compilateur Adam et pour métric mae (mean average error) et comme policy Epsilon Greedy pour le renforcement learning. Nous avons utilisé des paramètres car après des recherches ils semblaient les plus adaptés et standards dans la création de modèle de deep RL.

Les résultats peuvent être retrouvés plus bas et via l'interface graphique.

III. Sarsa

A. Définition

En apprentissage par renforcement, **SARSA** est un algorithme d'apprentissage. Son nom est l'acronyme de *State-Action-Reward-State-Action* (État-Action-Récompense-État-Action). C'est un algorithme on-policy : il utilise la politique en train d'être apprise pour mettre à jour les valeurs internes apprises.

Voici le pseudo-code de l'algorithme :

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

B. Réalisation

Pour appliquer les normes de cet algorithme, nous avons d'abord créé l'environnement en utilisant le libraire de GYM qui nous permet de rendre le jeu en plusieurs dimensions.

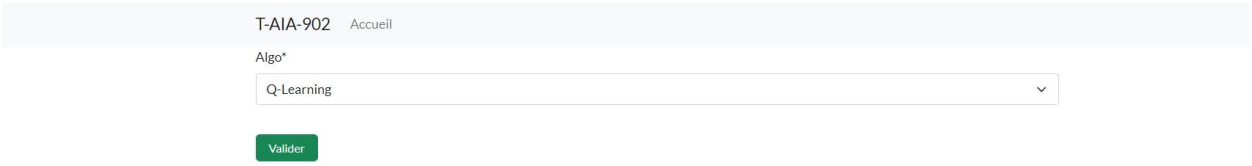
Ensuite, nous avons d'abord choisi les hyperparamètres qui nous permettent de régler les paramètres d'apprentissage pour l'entraînement pour que nous puissions avoir des résultats plus efficaces pendant les tests.

Finalement, nous avons développé le modèle dans lequel nous avons utilisé cet algorithme comme le cœur de fonctionnement du modèle pour choisir des bon pas qui permettent le modèle de prendre des décisions plus efficacement pour avoir des meilleurs rewards en jeu.

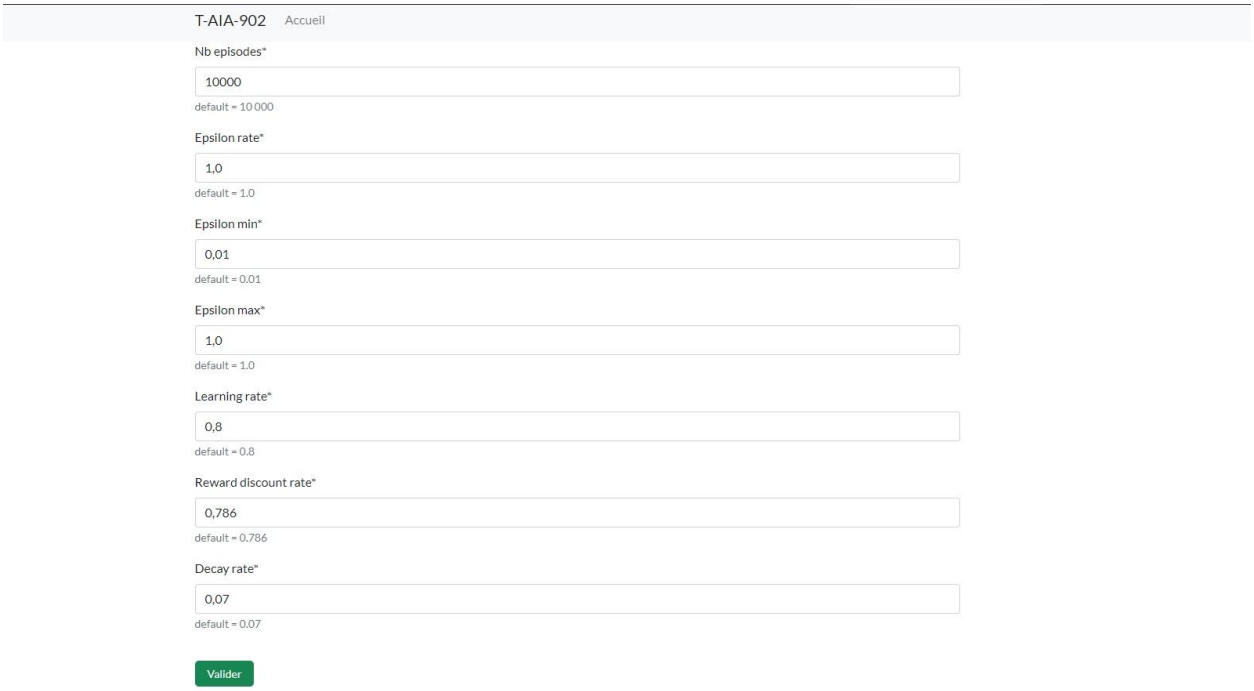
IV. Interface graphique

Afin de permettre à l'utilisateur de choisir son algorithme, et le personnaliser via les paramètres fournis, nous avons réalisé une web app avec django, avec 2 formulaires.

Le premier, permettant de choisir l'algo voulu, entre Sarsa, Q-Learning, et Deep Q-learning



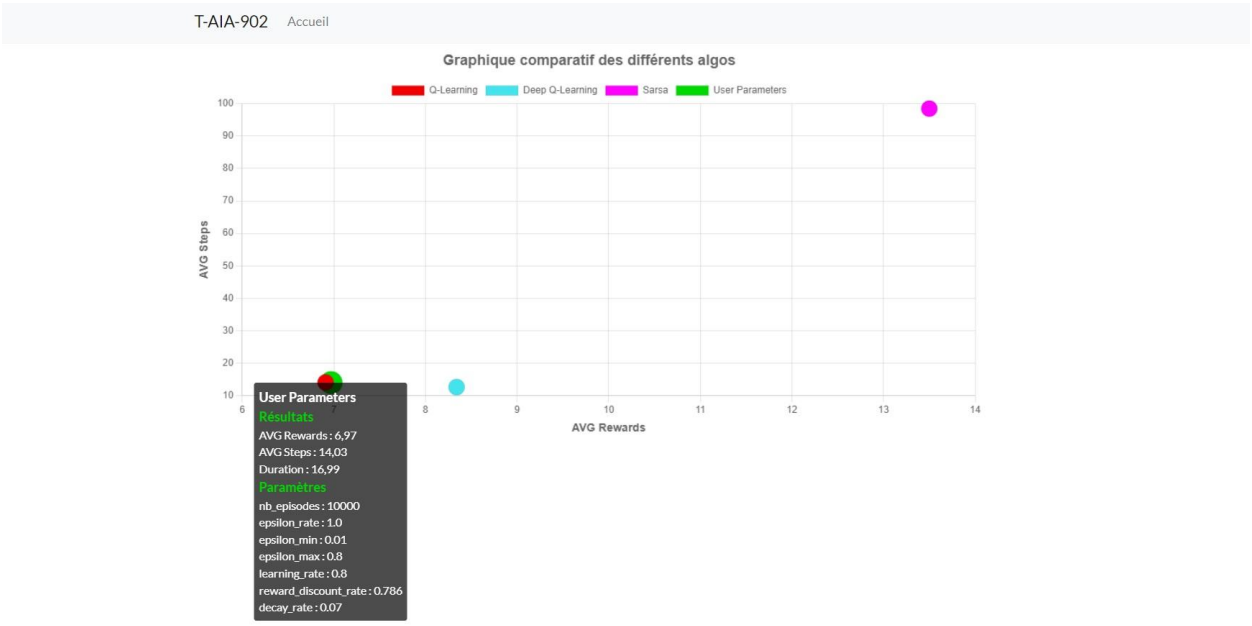
Le second, afin de choisir et modifier les paramètres associé à cet algo



Et enfin, une vue comparative des meilleurs résultats de nos 3 algos, ainsi que le résultat de l'algo choisi par l'utilisateur, avec ces paramètres.

Le comparatif du graphique se fait sur le nombre de step moyen réalisé afin de réussir le jeu, et la quantité moyenne de récompense reçue.

Au survole des points du graphique, des informations sont renseignées. Les paramètres entrés, la durée d'exécution, et les récompenses et step moyenne.


T-AIA-902
Accueil
Graphique comparatif des différents algos
Q-Learning
Deep Q-Learning
Sarsa
User Parameters
AVG Steps
AVG Rewards
User Parameters
Résultats
AVG Rewards : 6,97
AVG Steps : 14,03
Duration : 16,99
Paramètres
nb_episodes : 10000
epsilon_rate : 1.0
epsilon_min : 0.01
epsilon_max : 0.8
learning_rate : 0.8
reward_discount_rate : 0.786
decay_rate : 0.07