IAT : Space Invaders
-
Q-learning

Lina BELKACEM - Nicolas CAVALIER - Miguel DE OLIVEIRA - Luca FABRITUS - Titouan ISSARNI

3 juillet 2022

1 Introduction

Ce projet est une introduction à la résolution de problèmes en intelligence artificielle (IA) via l'apprentissage par renforcement. Le problème ici est d'entraîner une IA capable de jouer et de s'améliorer de manière autonome au jeu Space Invaders. Pour ce faire, nous avons choisi de résoudre ce problème avec une méthode d'apprentissage par renforcement nommé Q-learning. Pour pouvoir résoudre le problème à l'aide de cette méthode qui est tabulaire, nous avons dû définir les états de notre système et les réduire au plus pour pouvoir être efficace avec la méthode Q-Learning qui est très efficace sur un nombre d'états de taille finie. On expliquera alors dans un premier temps ce processus de définition d'états avant d'expliquer le fonctionnement du Q-learning. Enfin, on présentera l'optimisation que l'on a pu effectuer sur ce projet et on discutera de l'efficacité de notre algorithme et de notre agent.

2 Définition des états

2.1 Définition du problème

Le problème est un jeu de space invaders. Le jeu peut se représenter par une grille de 600×800 pixels. Il y a un vaisseau qui représente le joueur qui peut se déplacer uniquement de manière horizontale Et des invaders qui se déplacent horizontalement et à chaque fois qu'ils touchent une bordure ils se déplacent vers le bas. De plus le vaisseau peut tirer des projectiles pour abattre les invaders, ces projectiles se déplacent verticalement. Si nous utilisons cette vision pour définir notre système, le nombre d'états monte à 480000. On pourrait résoudre ce problème avec un tel nombre d'états, mais les phases d'apprentissage serait alors bien trop longue. On a alors dû réfléchir à un nouveau système d'états pour notre algorithme.

2.2 Définition des états pour le traitement

La difficulté est de trouver le bon compromis entre prévisions et nombre d'états pour traiter un problème d'apprentissage par renforcement. En effet, si on prend le problème global sans approximation, la précision sera importante, mais l'apprentissage trop long. À l'inverse, si on réduit trop notre système, alors l'apprentissage sera rapide et facile, mais les approximations seront trop grandes et le score de notre agent sera faible. On a alors choisi, plutôt que d'étudier l'image en entier et tous les paramètres qui en découlent, d'étudier les différences de positions entre notre vaisseau et les invaders. Ce qui nous intéresse maintenant pour résoudre le problème, c'est donc la position du vaisseau, la position des aliens, l'état du projectile du vaisseau (en train de tirer ou au repos) ainsi que l'action qui est choisie. Mathématiquement, on peut formaliser les états du problème de la manière suivante.

$$S = \{(x_v - x_i, y_v - y_i, s_b, a) \,|x_p, x_i \in [0, 800], y_p, y_i \in [0, 600], s_b \in [0, 1], a \in [0, 4]\}$$

avec x_v et y_v les positions horizontales et verticales du vaisseau, x_i et y_i les positions horizontales et verticales d'un invaders, s_b états du projectile 0 au repos et 1 en feu et a l'action qui est choisie.

3 Q-Learning

3.1 Comment ça marche

Le Q-learning est capable d'effectuer des actions aléatoires sans tenir compte de la politique actuelle. Cet algorithme cherche surtout à apprendre une politique qui maximise la récompense totale. Le Q-learning met en évidence l'utilité de l'action à accomplir pour obtenir une récompense. Nous avons décidé d'utiliser cet algorithme dans le cadre du projet. Comme tous les algorithmes d'apprentissage par renforcement, Q-Learning repose sur l'enchaînement de trois étapes principales : échantillonnage, apprentissage et amélioration.

3.2 Politique de l'agent

Pour apprendre à jouer, l'agent va devoir alterner entre exploration et exploitation. Quand une première simulation se lance, l'algorithme n'a pas de données pour faire son choix d'action, il va donc explorer des actions au hasard pour entamer la construction de sa matrice Q (fonction d'action-valeur). Dans le cas contraire, il optera pour l'exploration des données. Les taux d'exploration et d'exploitation sont définis par le paramètre epsilon qui sera abordé dans le paragraphe suivant. Au fur et à mesure que nous développons notre stratégie, nous avons moins besoin d'exploration et plus d'exploitation. Autrement dit, plus les essais augmentent, plus epsilon devrait diminuer. La matrice Q est notre point de départ pour déterminer les actions à suivre. Au fur et à mesure des actions, la matrice Q est mise à jour pour optimiser nos résultats selon la formule suivante vue en cours :

$$Q[etat][action]^{n+1} = (1-\alpha)[etat][action]^{n} + \alpha(reward + \gamma(Q[prochainetat]))$$

4 Hyperparamètrage et Optimisation

Dans l'apprentissage automatique, comme le Q-Learning, un hyperparamètre est un paramètre dont la valeur est fixée à l'avance et permet de contrôler le processus d'apprentissage de l'algorithme. Les différents hyperparamètres sont définis par la nature de l'algorithme d'apprentissage choisi, certains n'en nécessitent aucun et d'autres plusieurs. À partir de ces hyperparamètres, l'algorithme d'apprentissage apprend les paramètres à partir des données. Dans notre cas, à partir de notre modélisation du problème et de ceux du Q-Learning, nous avons six hyperparamètres :

— **Le pas d'apprentissage α :** Ce coefficient pondérateur va permettre d'apporter plus ou moins d'impacts dans le choix des actions futures en fonction de l'expérience des actions précédentes. Plus concrètement, fixer la valeur de α à 0 signifie que les valeurs de Q ne seront jamais mis à jour, donc rien ne sera appris. En revanche, en fixant une valeur de α proche de 1, cela signifie que l'apprentissage se produira rapidement.

— **L'importance donnée aux récompenses γ :** Les récompenses sont actualisées par un coefficient d'atténuation qui contrôle la balance entre l'importance des récompenses immédiates et futures.

— **Le temps d'exploration ϵ :** Epsilon-Greedy est une méthode simple pour équilibrer l'exploration et l'exploitation en choisissant entre l'exploration et l'exploitation au hasard. L'epsilon-gourmand, où epsilon fait référence à la probabilité de choisir d'explorer, exploite la plupart du temps avec une petite chance d'explorer. Epsilon est fixé à 1 au début de la simulation et diminuera progressivement de manière identique sur n épisodes puis sera constant sur les épisodes restants.

— **Le nombre d'épisodes :** Ce paramètre peut être comparé à une partie d'un jeu. Chaque épisode, ou partie, va permettre d'entraîner un agent. Chaque épisode représente une expérience complète qui prend fin pour l'une des trois raisons suivantes : un nombre maximum d'actions, l'agent a perdu, l'agent a achevé la tâche assignée.

— **Le nombre d'actions maximum par épisode :** Corresponds aux nombres d'actions maximums possibles par épisode pour éviter des épisodes infinis.

— **L'échantillonnage de la zone de jeu :** Plus spécifique à notre modélisation, afin de réduire le nombre d'états possibles, nous avons décidé d'appliquer un échantillonnage des zones de jeu pour localiser les Invaders. Cet échantillonnage s'applique sur les axes x et y réduit la précision du système mais permet un apprentissage plus important grâce à un nombre d'états possibles grandement réduit.

L'objectif est donc de trouver la combinaison de paramètres qui permet d'obtenir le meilleur score en réalisant des compromis sur les différents paramètres qui influencent le temps d'apprentissage, les ressources nécessaires, l'efficacité etc ... Ci-dessous, un tableau résumant les performances de chaque combinaison essayée :

Nombre épisodes	Max steps	Pas échantillonage	Gamma	Alpha	Eps profil*	Durée apprentissage	Score moyen
500	1500	1	0.95	0.1	1 - 0	51 minutes	116.9
500	1000	10	0.95	0.1	1 - 0	18 minutes	96.7
500	1000	1	0.95	0.1	1 - 0	28 minutes	61.2
500	1000	10	0.8	0.1	1 - 0	19 minutes	55.7
1000	1000	10	0.95	0.1	1 - 0	44 minutes	54.9
440	1000	50	0.95	0.1	1 - 0	13 minutes	49.6
500	1500	10	0.95	0.1	1 - 0	19 minutes	48.2
500	1500	10	0.95	0.3	1 - 0	33 minutes	26.3
500	1500	100	0.95	0.2	1 - 0	18 minutes	22.9
500	1000	100	0.95	0.3	1 - 0.1	11 minutes	18.5
1000	2000	50	0.95	0.1	1 - 0	1 heure et 7 minutes	16.7
500	1000	100	0.95	0.3	1 - 0	22 minutes	15.5
* --> eps_profil(initial, final)							

FIGURE 1 – Résultats en fonction des paramètres choisis

La principale contrainte pour déterminer les hyperparamètres est la suivante : aucun hyperparamètre n'est primordial par rapport à un autre. La solution optimale sera un couple de ces derniers. Les premiers hyperparamètres utilisés ont été définis de manière arbitraire, tout en utilisant un ordre de grandeur proche des valeurs vues durant les séances de TP. Par la suite, nous avons modifié presque tous les paramètres un à un. Par exemple, nous faisions varier le nombre d'épisodes vers une valeur supérieure et inférieure pour déterminer quel axe permettait d'obtenir un meilleur score.

5 Résultats finaux

À partir de la partie précédente, nous avons choisi de prendre les hyperparamètres suivants qui permettent d'obtenir un score moyen de 116.7 invaders tués (moyenne obtenue sur 10 parties) :

- **Le pas d'apprentissage α :** 0.1
- **L'importance donnée aux récompenses γ :** 0.95
- **Le temps d'exploration ϵ :** 1 - 0
- **Le nombre d'épisodes :** 500
- **Le nombre d'actions maximum par épisode :** 1500
- **L'échantillonnage de la zone de jeu :** 1

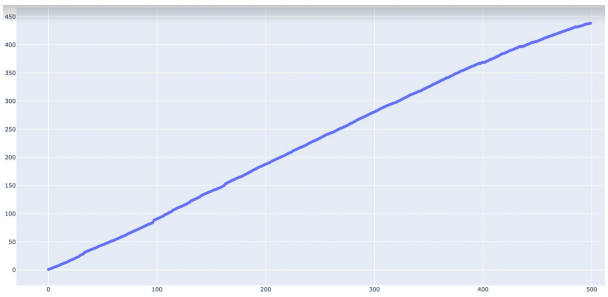

FIGURE 2 – Somme des valeurs de Q en fonction du nombre d'épisodes

Contrairement à ce que nous attendions, le Q Learning supporte un nombre d'états très grand : ici 800*600*2*4 = 3,84e6. Certes, le temps d'apprentissage doit être rallongé, mais les résultats obtenus sont meilleurs. D'ailleurs, on observe sur la figure ci-dessus, que l'apprentissage ne cesse de croître au cours des épisodes. On peut donc supposer que de meilleurs résultats auraient pu être obtenus en augmentant l'hyperparamètre max-steps. Par ailleurs on peut noter que la durée de l'apprentissage n'est pas garante d'un bon résultat, comme on peut le voir à l'avant dernière ligne du tableau.

6 Conclusion et axes d'améliorations

Pour conclure, les résultats à l'aide d'un algorithme de Q-Learning sont satisfaisants mais peuvent être améliorés avec des hyperparamètres mieux choisis. En effet, par le manque de temps pour réaliser le projet, nous n'avons pas pu mettre en place un script permettant de tester un très grand nombre de configurations et donc de choisir les valeurs les plus optimales. Avec les paramètres actuels, nous obtenons une moyenne de 96.7 invaders tués par partie avec 10 invaders sur l'écran en permanence. Les performances de notre programme sont conditionnées par le choix des états et des hyperparamètres. Il est toujours possible de les optimiser avec du temps.

La continuité du projet pour obtenir un meilleur score serait d'implémenter un algorithme de Deep Q-Learning qui permettrait alors de faire l'estimation de Q à l'aide d'un réseau de neurones. Cette méthode présente un inconvénient qui est le temps d'apprentissage qui devient très long en fonction de la complexité du problème.