IAT Rapport

Xinqing LI, Yuxuan SONG

April 2022

1 Introduction

Le but de ce projet est de réaliser un modèle d'apprentissage pour le jeu *Space Invaders*. Dans ce jeu, le joueur doit essayer d'attaquer des envahisseurs, puis gagne le point le plus haut possible. Notre IA joue un rôle comme le joueur, et il va apprendre l'expérience dans l'apprentissage de renforcement, puis tester sa performance finale.

2 Note

Notre groupe vous propose deux solution dans ce projet: l'algorithme Q-learning et l'algorithme DQN, mais en regard d'une meilleure performance, dans notre rapport, nous vous présentons notre l'algorithme Q-learning. Dans notre zip, nous vous proposons deux type d'algorithmes réalisés.

3 Tâches

3.1 Algorithme

Nous utilisons Q-learning dans ce projet. En mettant à jours le tableau-Q avec la fonction de Bellman, nous obtenons à la fin un tableau de valeurs correspondant aux actions dans chaque état. Parmi ces actions,ce qui possède une valeur la plus grande signifie un meilleur choix pour l'état à l'instant. Autrement dit, Tous les actions choisies en utilisant cet agent dépendent du tableau-Q dans cet algorithme.

Pourquoi nous ne choisissons pas DQN?
Nous avons aussi fait le coding en utilisant DQN, mais nous n'avons pas réussi et nous avons trouvé pas mal de problème à la fin.
L'avantage de DQN est de remplacer le tableau-Q par un réseau neurone. Cela épargne de l'espace pour sauvegarder des valeurs Q. Mais dans l'entraînement du réseau, nous avons le problèmes suivant: Pour DQN, nous avons un pool pour sauvegarder des expériences, mais la majorité de ces expériences sont avec la récompense nulle, car le nombre des états qui réussit à faire une attaque est vraiment rare, et quand nous essayons à faire un entraînement, c'est presque impossible de trouver une expérience avec une récompense de 1. Le conséquence de ce problème est que nous trouvons tous les valeurs de actions sont similaires pour un état, et le joueur répète la même action.

3.2 Formalisation

Dans un modèle de l'apprentissage de renforcement, des éléments doivent inclure des états, des actions, des récompenses, et des états prochains:
Définir les états:

- La position du joueur : player_X

- La position des aliens : invader_X, invader_Y
- L'état de bullet: bullet_state (1 signifie 'fire', et 0 signifie 'rest')

Définir les actions:

- Le joueur déplace vers la gauche: 0
- Le joueur déplace vers la droite:1
- Le joueur attaque: 2
- il n'y a pas d'action: 3

Définir les récompenses:

- Le joueur réussit à attaquer des envahisseurs: 1
- 0

Pour minimiser la taille de mémoire utilisée dans notre algorithme, spéciallement pour le tableau Q, nous divisons player_X par 10, invader_X et invader_Y par 20, dont la taille de notre tableau Q sera $80 \times 40 \times 30 \times 2 \times 4(player\_X \times invader\_X \times invader\_Y \times bullet\_state \times action)$.

3.3 Apprentissage/Hyperparamétrage

3.3.1 Hyperparamétrage

Dans notre algorithme, nous allons définir plusieurs hyperparamétrages avec des usages différents.
Premièrement, dans chaque algorithme d'apprentissage, quelques soient l'apprentissage supervisée, l'apprentissage non supervisée, taux d'apprentissage (Learning Rate, LR) est un élément nécessaire. Ici, nous supposons $LR = \alpha$
Pour sélectionner une action pour chaque état, nous utilisons le stratégigreedy-selection, avec un taux de choix aléatoire ϵ. Pour exploiter plusieurs possibilités, ϵ sera grand au début, et petit à la fin. Donc ici, nous avons trois paramètres: init_eps, final_eps, et après chaque épisode, $self.epsilon = max(self.epsilon - 1./(n\_episodes - 30.), self.eps\_profile.final)$
Le coeur de Q-learning est $\delta \leftarrow r + \gamma max_{a' \in A} Q(S', a') - Q(s, a)$, qui renouvelle des valeur de Q. Ici, représente le taux de long voir, c'est-à-dire le pouvoir de l'influence de futur à cet état.
Dans l'application numérique, nous prenons:

	α	0.2
	γ	1.0
ϵ	init_eps	1.0
ϵ	final_eps	0.1
	$\max_s tep$	100000
	$\max_e pisodes$	100

Table 1: Hyperparamètres

3.3.2 QAgent Class

Nous avons 4 parties dans cette classe: select_action(state), updateQ(state, action, reward, next_state) et learn().

- select_action(state): Nous utilisons le stratégie greedy-selection pour explorer plusieurs possibilités dans l'environnement. Dans ce cas, une valeur aléatoire est choisi pour comparer avec la valeur *epsilon* , si elle est supérieur à ϵ , nous choisissons le meilleur choix (aussi une action qui possède la plus grande valeur Q) selon le tableau Q, sinon nous choisissons une action aléatoire.

- updateQ(state, action, reward, next_state): Nous calculons la valeur de Q en utilisant notre formule avant et nous mettons à jour cette valeur Q:

$$Q[state][action] \leftarrow (1-\alpha)Q[state][action] + \alpha(reward + \gamma max(Q[next\_state])$$

- learn(): Nous faisons l'entraînement d'un nombre d'épisodes 200, et dans chaque épisode, nous commençons un nouveau jeux par env.reset(). Le nombre d'étape dans chaque épisode est limité à 100000 étapes. Dans chaque étape, nous prenons les différentes actions par select_action(state), et joue avec cette action (env.step(action)). env.step(action) va nous tourne le prochain état, la récompense, et si le jeu est fini ou pas. Nous utilisons ces informations à modifier notre tableau Q dans updateQ(state, action, reward, next_state). Après chaque étape et chaque épisode, notre tableau converge la performance.

3.4 Analyse des résultats

Nous réussissons de démarrer notre jeu en utilisant QAgent mais la performance n'est pas idéale.

Problèmes possible:

- Délai du temps. Dans ce jeu, il y a un délai entre l'action attaque et la récompense 1. À cause de cette raison, la liaison entre les états qui réussi l'attaque et leur récompense 1 est faible. Donc la valeur de la récompense correspondant à l'état de ce temps n'est pas exactement propre.
- Les valeurs de hyperparamètre. Comme les hyperparamètres que nous utilisons dans le formule pour calculer la valeur Q effectuer également le résultat, il faut les modifier et essayer plusieurs fois les différentes valeurs pour trouver la meilleur performance.

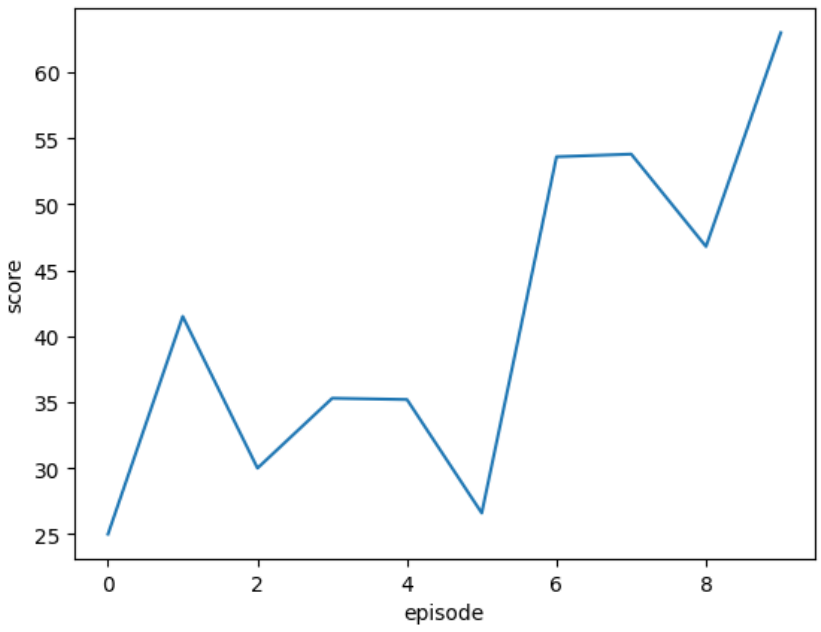

Figure 1: Score QAgent,moyenne de 10 épisodes par un point

D'après la figure, la performance à la fin n'est pas trop mal, même si elle ne converge pas à la fin du épisode dans la figure, nous croyons après le temps longues la performance peut converger.

Résultat du jeu

- En épisode 60, la valeur ϵ converge à 0.1
- Pour 100,000 étapes nous avons le score 113.

4 Conclusion

La performance de notre QAgent n'est pas trop idéal maintenant, dans l'étude futur nous allons essayer de converger la performance plus vite possible.