Suika Game

Création d'un agent autonome - Hugo Monchal

Introduction

Suika Game (ou jeu de la pastèque) est un jeu arrivé en France en 2023. Très vite devenu populaire, ce jeu consiste à laisser tomber des fruits dans un bac. Lorsque deux fruits similaires entrent en contact, ils fusionnent en un fruit plus grand et le joueur gagne des points.

Le but du jeu est de faire un maximum de points en faisant fusionner les fruits jusqu'à obtenir la pastèque qui est le plus gros fruit possible. La partie s'arrête lorsqu'un fruit sort du bac par manque de place.

Mon projet à donc pour but d'entraîner un agent à jouer à ce jeu, et qu'il soit le meilleur possible.

Fonctionnement de l'agent

- Il **récupère l'état du jeu (St)** (image des fruits présents dans le bac, fruit à placer ce tour ci, fruit à placer au prochain tour)
- Grâce à un CNN qui prend en entrée l'état du jeu, il **prédit une valeur Q** qui correspond à une estimation de la récompense **pour chaque action possible**. Ici, j'ai choisi de diviser la largeur du bac (qui est un espace continu) en un espace discret de taille A. La valeur **Qπ(St,At)** tente de prédire la récompense perçue sachant l'état du jeu, si l'action a est réalisée.
- L'**action** à effectuer sera donc l'action pour laquelle la valeur Q prédite est la plus grande.
- Son action va mener à un nouvel état du jeu (**St+1**), et lui donner une **récompense (Rt+1)**, par exemple en fonction du score engendré par son action. Il va enregistrer toutes ces variables dans sa mémoire pour chaque coup.
- Après un certain nombre de coups joués, il va s'**entraîner** et mettre à jour les poids de son réseau en minimisant la MSE entre la récompense prédite et la récompense perçue. La récompense perçue va être modifiée en prenant en compte la récompense prédite du prochain coups, afin de maximiser les récompense sur le long terme.

$$Q\pi(St,At) \leftarrow Q\pi(St,At) + yQ\pi(St+1,At+1)])$$

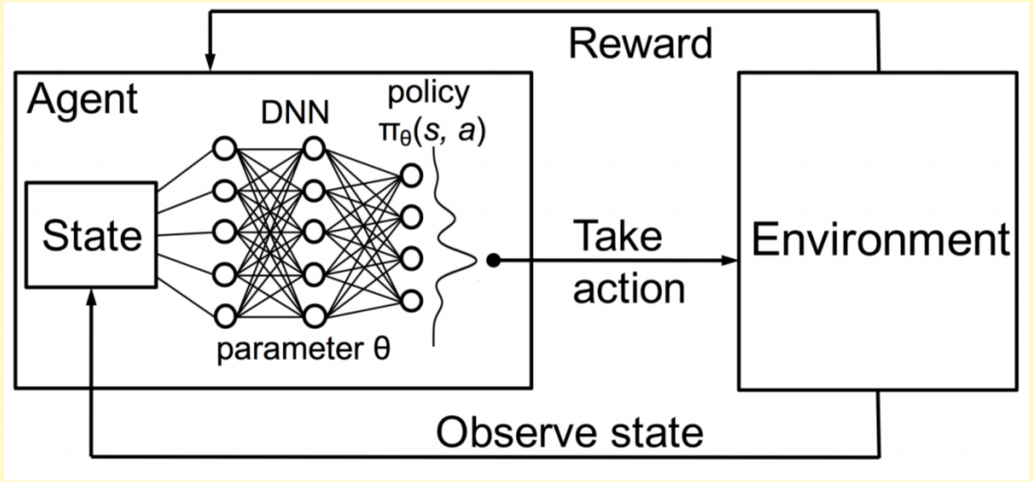
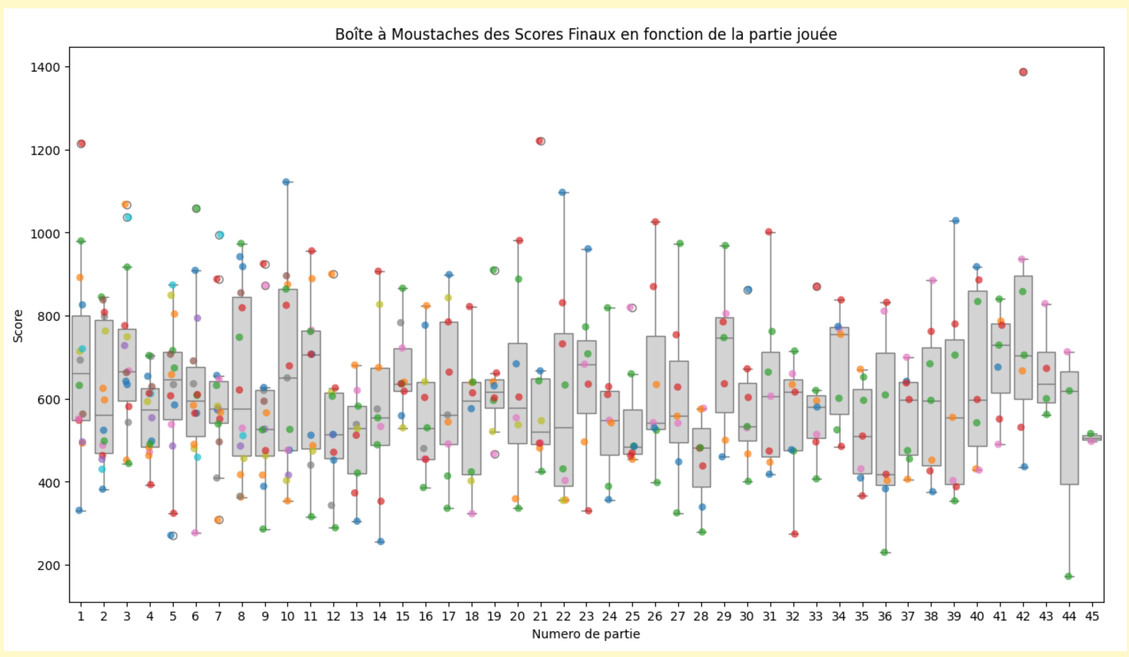


Schéma DQN [4]

Résultats des agents DQN



Les agents jouent aléatoirement dans leurs premières parties, c'est la phase d'exploration. On constate néanmoins qu'après plusieurs parties, au cours desquelles ils ont pratiqué plusieurs phases d'entraînement, qu'il n'y a pas d'amélioration significative de leurs scores.

Malgré plusieurs tentatives avec des réseaux de neurones différents, ou une politique de récompense différente, je n'ai pas réussi à créer un agent meilleur que le hasard.

Il est probable que l'espace dans lequel il doit choisir son action soit trop grand, ou que le CNN n'arrive pas à détecter suffisamment d'informations sur l'image du plateau de jeu.

Méthodologie

J'ai dans un premier temps récupéré une version du jeu implémentée en PyGame par Ole-Batting [1] dans le but de pouvoir simuler en locale autant de parties que je veux, et de pouvoir faciliter les interactions de mon agent avec le jeu.

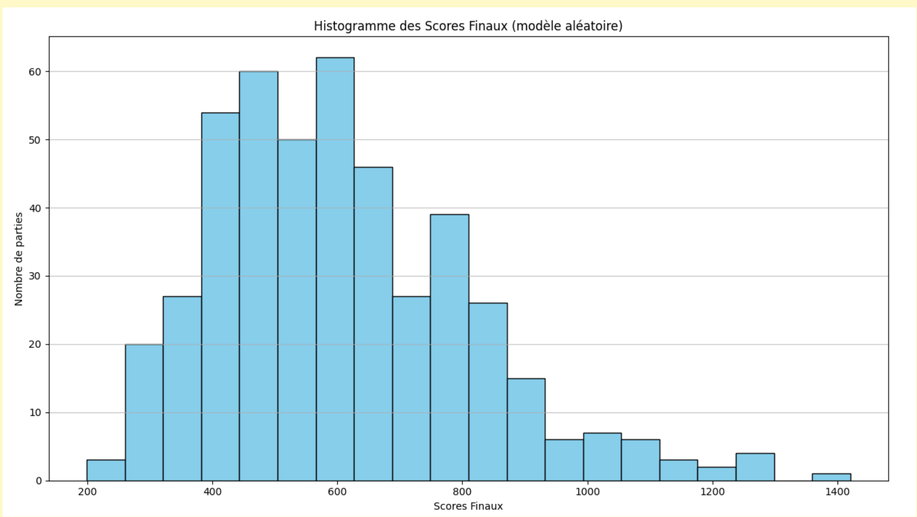
Après quelques modifications, J'ai simulé un agent qui joue aléatoirement afin d'avoir un référentiel pour comparer les performances d'un agent entraîné.

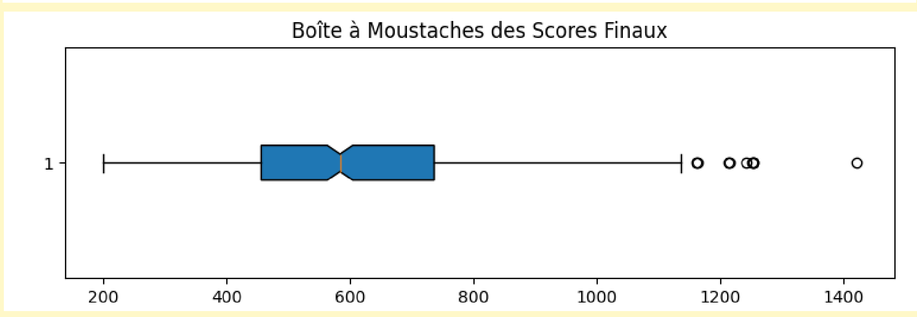
Ensuite, j'ai créé un agent capable d'apprendre, basé sur un Deep Q Network [2][3], qui est une méthode efficace d'apprentissage par renforcement.

Enfin j'ai comparé mon agent avec le hasard pour voir s'il faisait mieux que le hasard

Agent aléatoire

- Moyenne des scores finaux : 606
- Écart-type des scores finaux : 204





Conclusion

Les agents DQN n'ont pas réussi à faire mieux que le hasard. Cette approche n'est sans doute pas la plus adaptée à mon problème. Elle semble être peu efficace dans des environnements complexes, et surtout lorsque l'espace des actions est aussi grand.

Il existe de nombreuses approches similaires, plus efficaces dans le cadre d'un espace d'action continu, telles que le Deep Deterministic Policy Gradient (DDPG) ou le Soft Actor-Critic (SAC). De plus, on pourrait appliquer un apprentissage par algorithme génétique, afin de sélectionner les meilleurs agents pour aboutir plus rapidement à un agent fort au jeu.

Néanmoins par manque de temps et à cause des difficultés rencontrés en implémentant mon agent, je n'ai pas pu tester ces autres approches qui sont également plus complexes. Au vu des résultats de mes agents, la sélection d'agents dans l'algorithme génétique n'aurait pas abouti à de meilleurs scores.

Ce fut malgré tout une bonne expérience qui m'a permit d'en apprendre plus sur l'apprentissage par renforcement, et plus particulièrement sur le DQN

Références:

- [1] **Ole-Batting**, Suika game Pygame implementation, *https://github.com/Ole-Batting/suika*
- [2] **Lukas Schwarz**, *Explanation and Implementation of DQN with Tensorflow and Keras*, https://lukasschwarz.de/dqn
- [3] **Amrani Amine**, *Deep Q-Networks: from theory to implementation*, https://towardsdatascience.com/deep-q-networks-theory-and-implementation-37543f60dd67
- [4] **莉森揪** , *Deep Q Network*, https://ithelp.ithome.com.tw/articles/10208668

Code du projet disponible sur **Github**:
https://github.com/hugoImonchal/suika_agent.git