Hugo DENIS–MARTIN
Brice MABILLE

Rapport Reinforcement Learning

Introduction

Ce projet applique des techniques de Reinforcement Learning au jeu de Blackjack, un problème classique de prise de décision. L'objectif est d'explorer les performances de différents agents dans une version simplifiée du jeu.

Ce projet permet d'analyser et de comparer leurs comportements, tout en offrant des insights sur l'influence des paramètres d'apprentissage.

Définition de l'Environnement

Les principales règles du jeu ont été implémentées depuis le code python "Jeu_regles" :

Objectif : L'objectif du joueur est de battre le croupier en obtenant une main dont la valeur totale est plus proche de 21 sans dépasser cette limite.

- **Valeur des cartes** :

Les cartes numérotées (2 à 10) conservent leur valeur nominale. Les figures (Roi, Dame, Valet) valent 10 points. L'As peut valoir soit 1, soit 11, selon ce qui est le plus avantageux pour le joueur.

- **Actions possibles** :

Hit : Tirer une carte et augmenter la valeur de la main. - **Stand** : Conserver sa main actuelle.

- **Règles du croupier** :

Le croupier doit tirer des cartes jusqu'à ce que la valeur totale de sa main atteigne ou dépasse 17.

Ces règles définissent un environnement qui permet de tester et d'analyser différents agents d'apprentissage. L'utilisateur peut jouer au BlackJack en exécutant le code "Jeu_blackjack.py".

Agent Aléatoire

Cet agent a été créé dans le code "Agent_random.py". Cet agent ne tient pas compte de la valeur de sa main ni des cartes visibles du croupier. Il nous est utile pour pouvoir comparer les performances des autres agents mais ne possède aucune stratégie.

Remarque : Cet agent illustre un comportement totalement exploratoire, sans exploitation d'informations.

Agent Simple

Cet agent a été créé dans le code "Agent_simple.py".

Stratégie : Lorsque le joueur reçoit ces 2 cartes initiales, l'agent décide selon la valeur de la main s'il dépasse ou non le résultat 17. Si la valeur de la main est inférieure à 17, l'agent tire une carte (*hit*). Si la valeur est 17 ou plus, l'agent s'arrête (*stand*).

Cependant, cet agent ne prend pas en compte les cartes visibles du croupier ni l'historique des tirages. Ces performances sont donc limitées face à des stratégies optimisées.

Q_learning Epsilon-Greedy

Cet agent a été créé dans le code "Agent_Q_learning.py".

Stratégie : Lorsque l'agent Q-learning reçoit un état initial, il décide de l'action à entreprendre en fonction de la valeur de l'état actuel et de la politique apprise. Après chaque action, l'agent met à jour ses valeurs de Q en fonction de la récompense reçue et de la future récompense anticipée, selon la formule de mise à jour de Q-learning. Cette approche permet à l'agent d'adapter ses décisions au fur et à mesure de l'apprentissage.

Cependant, malgré sa capacité d'apprentissage, cet agent reste limité par l'exploration de son espace d'actions et peut ne pas converger vers la politique optimale dans cet environnement complexe. Nous avons donc varier les paramètres de l'agent Qlearning pour optimiser au mieux exploration et exploitation.

SARSA

Cet agent a été créé dans le code "Agent_Sarsa.py".

Stratégie : Contrairement au Q-learning, où l'agent met à jour sa valeur d'action en utilisant la meilleure action possible (celle qui maximise la récompense future), l'agent SARSA met à jour sa fonction Q en utilisant l'action qu'il a réellement choisie, ce qui reflète une politique on-policy. Cela signifie que les mises à jour de la fonction Q dans SARSA prennent en compte l'exploration effectuée pendant l'apprentissage, plutôt que d'opter pour l'action la plus optimiste comme dans Q-learning.

Cela permet à l'agent SARSA d'apprendre de manière plus prudente dans des environnements où l'exploration est plus risquée.

Comparaison

Nous avons ensuite comparé les agents pour comprendre la meilleure stratégie à adopter. L'agent aléatoire nous renvoie des résultats imprévisibles, il a 30% de chance de gagner face au dealer. L'agent simple ne tient pas compte des optimisations possibles, mais avec une condition supérieure à 17, il obtient 42% de chances de gagner. Les agents Q-learning et SARSA apprennent progressivement mais Q-learning maximise les récompenses futures de manière optimiste, tandis que SARSA met à jour ses valeurs en fonction des actions réelles choisies, ce qui est plus prudent.