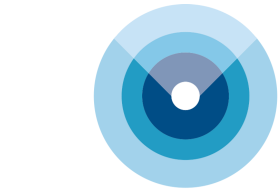

Université Côte d'Azur

Master Informatique, parcours Intelligence Artificielle

Project report

Apprentissage par Renforcement sur Mario Bros

Réalisé par :
DE SEROUX Colin & HADDOU Amine

Encadré par :
MARTINET Jean

Le :
21 octobre 2024

Apprentissage par Renforcement

Mario Bros

21/10/2024

Table des matières

1 Introduction

L'objectif de ce projet est de développer un modèle d'apprentissage par renforcement appliqué au contexte des jeux vidéo. Nous avons choisi d'utiliser le célèbre jeu *Super Mario Bros* comme environnement d'entraînement pour notre modèle. L'idée est de faire en sorte que le modèle puisse apprendre à terminer un niveau personnalisé en franchissant des obstacles tout au long du parcours.

2 Description de l'environnement

Le modèle doit parcourir une carte personnalisée comprenant des obstacles tels que des trous, des blocs qui obstruent le passage et d'autres qui empêchent les sauts. Le modèle dispose de deux actions possibles : avancer ou sauter. L'action "sauter" permet au modèle de se déplacer de deux cases en avant, en sautant au-dessus de la case adjacente. Vous trouverez le détaille des actions ici.

3 Algorithmes d'apprentissage

Pour entraîner notre modèle, nous avons décidé d'explorer deux types d'algorithmes d'apprentissage par renforcement : *Q-Learning* et *Monte Carlo Tree Search* (MCTS). Dans ce rapport, nous détaillerons dans un premier temps la mise en œuvre du *Q-Learning*, tandis que la partie concernant *MCTS* sera traitée ultérieurement.

3.1 Q-Learning

Le *Q-Learning* est un algorithme d'apprentissage par renforcement hors-ligne qui vise à apprendre une politique optimale pour maximiser le cumul des récompenses au fil du temps. Dans notre cas, nous avons paramétré le modèle de la manière suivante :

- **Epsilon-greedy strategy :** Nous avons initialisé ϵ à 1 avec un taux de décroissance de 0.001. À chaque épisode, ϵ est mis à jour selon la formule suivante :

  $$\epsilon = (\text{episodes} - \text{episode}) \times \text{decay}$$

  Cela permet de s'assurer que l'exploration est dominante au début de l'entraînement, tandis que l'exploitation devient progressivement plus importante en fin d'apprentissage.

- **Fonction de récompense :** Un trou sur le parcours fait perdre 100 points (-100), tandis que terminer un niveau rapporte 100 points (états finaux). Il est également possible de collecter des pièces qui rapportent 5 points. À chaque pas, une pénalité de -1 est appliquée pour inciter le modèle à optimiser ses mouvements et terminer le niveau plus rapidement (300 épisodes pour un modèle finissant le jeu de manière poussive).

- **Paramètres d'apprentissage :** Nous avons choisi un taux d'apprentissage de 0.1 et un facteur de discount (γ) de 0.99 pour mettre l'accent sur les récompenses futures.

3.2 MCTS

L'algorithme *MCTS* (Monte Carlo Tree Search) est une méthode d'exploration efficace pour les jeux et les problèmes de décision. Il combine la recherche aléatoire et l'optimisation des décisions, ce qui le rend particulièrement adapté aux environnements complexes avec de nombreuses possibilités d'actions. L'algorithme suit quatre phases principales : **séléction**, **expansion**, **simulation** et **backpropagation**. En construisant un arbre de décision basé sur des simulations de jeux, *MCTS* permet de déterminer la meilleure action à entreprendre, en tenant compte des résultats passés.

- Dans le contexte de mon jeu, où le joueur se déplace sur une grille avec comme actions **avancer** ou **sauter**, l'algorithme *MCTS* joue un rôle clé dans la prise de décision. Au début de chaque tour, *MCTS* exécute plusieurs itérations pour simuler des mouvements possibles à partir de l'état actuel du jeu. Chaque nœud de l'arbre représente un état spécifique du jeu, et les branches représentent les actions réalisables. Ce processus permet à l'algorithme d'explorer à la fois des mouvements connus et de nouveaux mouvements, en s'assurant d'optimiser les décisions en fonction des résultats des simulations.

- Le fonctionnement de *MCTS* peut être divisé en quatre phases. **La sélection** consiste à parcourir l'arbre des états jusqu'à atteindre un nœud non complètement exploré ou une feuille. **L'expansion** ajoute un nouveau nœud à l'arbre pour un mouvement non encore exploré. **La simulation** implique de jouer une série de mouvements aléatoires jusqu'à la fin de la partie, permettant d'estimer le résultat potentiel de cet état. Enfin, **la backpropagation** ajuste les valeurs des nœuds parent en fonction du résultat de la simulation, améliorant ainsi la précision des estimations pour les décisions futures.

- L'efficacité de *MCTS* repose sur l'équilibre entre exploration et exploitation. L'algorithme utilise une stratégie appelée UCT (Upper Confidence Bound for Trees) pour choisir les mouvements, favorisant les actions avec un faible nombre de visites tout en tenant compte de leur score moyen. Cela permet à *MCTS* de découvrir de nouvelles stratégies tout en optimisant les meilleures options déjà identifiées. En somme, *MCTS*, en intégrant ces mécanismes d'exploration, améliore considérablement la capacité du joueur à prendre des décisions éclairées, ce qui se traduit par des performances optimales dans le jeu.

- De plus, pour améliorer la recherche et la rapidité, j'ai fait en sorte que, non seulement le score final, les trous ou les pièces soient pris en compte, mais aussi le fait d'avancer. Le reward se calcule donc par :

$$\text{reward} = \text{score} + \frac{\text{player\_pos}[0] - \text{start\_pos}[0]}{100}$$

  ce qui permet d'obtenir de meilleurs résultats avec un nombre d'itérations plus faible, vu qu'il va toujours essayer d'avancer (il va surtout beaucoup sauter, vu que cela fait 2 déplacements en 1 action dans certaines configurations. Il en est de même pour le *Q-Learning*).

4 Résultats et Conclusion

L'implémentation de notre modèle d'apprentissage par renforcement à l'aide de l'algorithme *Q-Learning* a été couronnée de succès. Le modèle a réussi à apprendre efficacement à éviter les obstacles et à collecter les pièces tout en atteignant l'état final, qui est la fin du niveau. L'utilisation de la stratégie ϵ-greedy a permis un équilibre adéquat entre exploration et exploitation au cours de l'entraînement, tandis que la fonction de récompense et les paramètres d'apprentissage ont favorisé une convergence rapide vers une politique optimale.

De même, l'algorithme *MCTS* parvient sans difficulté à terminer la partie en récupérant les pièces sans tomber dans les trous. De plus, *MCTS*, contrairement au *Q-Learning*, n'a pas besoin d'entraînement, ce qui permet de le déployer directement sur d'autres nouvelles cartes.

Dans les deux cas, les algorithmes sont une réussite, avec un temps d'entraînement inférieur à 1 seconde pour le *Q-Learning* sur une carte de plus de 45 blocs de large, et moins de 0,1 seconde pour chaque coup pour *MCTS*.