Université Sorbonne
Faculté des Sciences et Ingénierie
Licence Informatique - Troisième Année

Projet de Recherche

Reinforcement Learning

Rayane Yadel
Rayane Nasri
Isaac Kinane

Encadré par
Nicolas Baskiotis
Jean-Noël Vittaut

15 mars 2025

1 Résumé

Ce projet vise à analyser et comparer différentes approches algorithmiques pour la création de bots dans des environnements de jeux vidéo. Nous avons implémenté et évalué deux méthodes principales : un algorithme d'apprentissage par renforcement, le Q-Learning et une approche par recherche arborescente via le Monte Carlo Tree Search (MCTS). Ces algorithmes ont été testés sur deux simulateurs distincts, à savoir le Tic-Tac-Toe, un jeu de plateau simple, et le Mad Pod Racing, un jeu d'action nécessitant la discrétisation d'un espace d'états continu. Les résultats préliminaires montrent une convergence vers une politique optimale pour le Q-Learning, tout en révélant des limitations liées à la dépendance du simulateur et aux fluctuations de performance. Nous n'avons pas encore réussi à apporter une solution efficace au problème du Tic Tac Toe Ultimate. La suite du projet s'orientera vers le raffinement de ces approches et l'exploration d'algorithmes complémentaires pour répondre aux problématiques soulevées.

2 Introduction

Au cours des dernières années, avec l'émergence d'AlphaGo, l'apprentissage par renforcement est devenu l'un des domaines les plus prometteurs pour le développement de bots destinés aux jeux vidéo. Toutefois, dans de nombreuses situations, recourir à des algorithmes classiques d'exploration, qu'ils soient stochastiques ou non, peut s'avérer plus simple et suffisant. Ces algorithmes, comme le Monte Carlo Tree Search (MCTS), présentent l'avantage d'être génériques et capables de s'adapter à une large variété de jeux. L'objectif du projet est d'analyser et de comparer les performances de diverses catégories d'algorithmes pour la création de bots, en tenant compte de leur caractère générique et de leur capacité à être adaptés à différents types de jeux.

Dans cette première phase du projet, nous avons commencé par une étude du cadre théorique de ce nouveau paradigme d'apprentissage, en nous appuyant sur l'ouvrage de Richard S. Sutton et Andrew G. Barto intitulé *Reinforcement Learning : An Introduction*. Par ailleurs, nous avons implémenté deux algorithmes de création de bots : un algorithme classique d'exploration, le Monte Carlo Tree Search, et un algorithme d'apprentissage par renforcement, le Q-learning que nous allons présenter par la suite.

Nous avons choisi deux jeux sur lesquels appliquer nos algorithmes : le premier est un jeu de plateau, le Tic Tac Toe Ultimate, et le second est un jeu d'action, le Mad Pod Racing.

3 Apprentissage par renforcement

Comme défini dans [1], l'apprentissage par renforcement est une approche computationnelle de l'apprentissage fondée sur l'interaction. Il s'agit d'apprendre quelles actions entreprendre pour maximiser un signal de récompense numérique. L'agent, ou apprenant, n'est pas dirigé sur les actions à choisir, mais doit plutôt les explorer afin de découvrir celles qui génèrent la récompense la plus élevée. Il convient de retenir les concepts clés suivants : *agent, récompense, action, environnement et découverte (exploration).*
L'apprentissage par renforcement présente de nombreuses similitudes avec l'apprentissage humain, voire animal. Prenons l'exemple d'un étudiant préparant ses examens : en résolvant des annales, il a tendance à **exploiter** les connaissances acquises lors des séances de TD. Toutefois, il lui arrive aussi d'essayer de nouvelles approches dans le but d'aller plus vite et d'obtenir une meilleure note, soit une récompense plus élevée. De manière similaire, un agent agit de la même

façon : parfois, il **exploite** ce qu'il a déjà appris, et parfois, il **explore** de nouvelles situations. C'est un problème que nous aborderons ensuite : celui de l'équilibre entre l'exploitation et l'exploration.

La modélisation mathématique du problème de l'apprentissage par renforcement repose sur la théorie des systèmes dynamiques et se formalise plus précisément comme un problème de contrôle optimal appliqué à des processus de décision markoviens partiellement connus. Ce point ne sera pas développé dans le présent écrit, car il dépasse le cadre de nos travaux ; toutefois, une formalisation complète et rigoureuse est disponible dans [1].

3.1 L'algorithme QLearning

3.1.1 Formalisation de l'algorithme

L'apprentissage par renforcement se divise en méthodes basées sur la valeur et celles basées sur la politique. Nous rappelons brièvement les concepts clés :

Définition 1 (**Politique**)**.** La politique π détermine, pour un état s, la probabilité de choisir une action a, soit de manière déterministe, soit stochastique via $\pi(a \mid s)$.

Définition 2 (**Récompense**)**.** La récompense $R_t = r(s, a, s')$ est le signal numérique reçu lors de la transition de s à s' par l'action a, indiquant la qualité de cette transition.

Définition 3 (**Retour**)**.** Le retour G_t est la somme actualisée des récompenses futures :

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}.$$

Il mesure le gain cumulé attendu.

Définition 4 (**Valeur d'un état**)**.** La valeur d'un état est l'espérance du retour lorsqu'on suit une politique π :

$$V^\pi(s) = \mathbb{E}[G_t \mid s, \pi].$$

Définition 5 (**Valeur d'une action**)**.** La q-value $Q^\pi(s, a)$ est l'espérance du retour après avoir exécuté l'action a dans l'état s, puis suivi la politique π :

$$Q^\pi(s, a) = \sum_{s', r} p(s', r \mid s, a)\Big[r + \gamma V^\pi(s')\Big].$$

Les méthodes basées sur la valeur estiment ces fonctions afin d'extraire une politique optimale, tandis que celles fondées sur la politique les apprennent directement. Le Q-Learning, approche emblématique, repose sur la mise à jour itérative de la Q-Table à l'aide de l'équation de Bellman :

$$Q(s, a) \leftarrow Q(s, a) + \alpha\Big[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\Big],$$

où α est le taux d'apprentissage. L'agent commence par explorer l'environnement (avec une stratégie ϵ-greedy) et met à jour la Q-Table au fil des épisodes, jusqu'à convergence.

Nous présentons ci-dessous l'algorithme formel du Q-Learning :

Algorithm 1 QLearning

Require: α : taux d'apprentissage, γ : facteur d'actualisation, ϵ : probabilité d'exploration, N : nombre d'épisodes

Require: $Q(s, a)$ initialisé arbitrairement

for épisode = 1 à N **do**
  Initialiser l'état s
  while s n'est pas terminal **do**
    Choisir une action a selon une politique ϵ-gloutonne basée sur $Q(s, a)$
    Exécuter l'action a et observer r et s'
    $Q_{\text{observée}}(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$
    $TD\_Error \leftarrow Q_{\text{observée}}(s, a) - Q(s, a)$
    $Q(s, a) \leftarrow Q(s, a) + \alpha \cdot TD\_Error$
    $s \leftarrow s'$
  end while
end for

3.1.2 Équilibre entre exploration et exploitation

L'un des défis majeurs de l'apprentissage par renforcement consiste à trouver un équilibre entre exploration et exploitation [1]. L'agent doit exploiter ses expériences passées pour maximiser les récompenses immédiates, tout en testant de nouvelles actions pour découvrir de meilleures stratégies. Une focalisation exclusive sur l'une ou l'autre de ces approches conduit à un apprentissage sous-optimal, voire à l'échec.

Bien que plusieurs techniques avancées existent pour optimiser cet équilibre, notre implémentation adopte une stratégie simplifiée : une politique ϵ-greedy. Ainsi, à chaque étape, l'agent choisit une action aléatoirement avec une probabilité ϵ (exploration), ou sélectionne l'action optimale selon sa fonction de valeur avec une probabilité $1 - \epsilon$ (exploitation).

3.1.3 Application à Tic Tac Toe et Mad Pod Racing

Tic Tac Toe Chaque configuration de la grille constitue un état s, et chaque coup valide (placer un 'X' ou un 'O') est une action a. La Q-Table, de dimension $3^9 \times 9$, reste exploitable pour ce jeu. Nous avons développé un simulateur et entraîné un agent Q-Learning avec :

— Politique ϵ-greedy avec $\epsilon = 0.5$,
— Facteur d'actualisation $\gamma = 0.9$,
— Taux d'apprentissage $\alpha = 0.1$.

L'agent reçoit une récompense de $+1$ en cas de victoire, -1 en cas de défaite et 0 pour un état non terminal. L'agent joue contre un adversaire aléatoire. La *Figure 1* illustre l'évolution du pourcentage de victoires et de parties non perdues, convergeant vers 100% (un joueur optimal ne perd jamais au Tic Tac Toe).

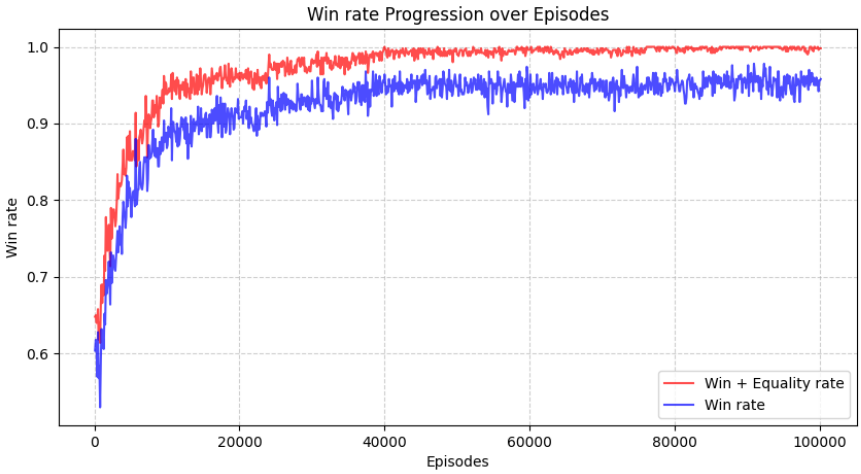

FIGURE 1 – Évolution du pourcentage de victoires et de non-défaites en fonction du nombre d'itérations.

Mad Pod Racing Dans Mad Pod Racing, un état initial est défini par le tuple $(x, y, v_x, v_y, \alpha, x_c, y_c)$ où :

- (x, y) sont les coordonnées du bot,
- (v_x, v_y) ses composantes de vitesse,
- α l'angle d'orientation,
- (x_c, y_c) les coordonnées du prochain checkpoint.

Cet espace d'états continu est discrétisé selon :

$$d = \sqrt{(x_c - x)^2 + (y_c - y)^2}$$

pour la distance, répartie en quatre intervalles : $[0, 700]$, $]700, 900]$, $]900, 1200]$, $]1200, +\infty[$, et

$$\beta = \left| \left((\arctan \frac{y_c - y}{x_c - x} \mod 360) - \alpha + 180 \right) \mod 360 - 180 \right|$$

pour la différence angulaire, discrétisée en $[0, 10]$, $]10, 45]$, $]45, 90]$, $]90, 180]$.

Les actions possibles se limitent aux accélérations $\{1, 25, 50, 75, 100\}$, auxquelles est ajouté un booléen indiquant l'activation du boost. Ainsi, la Q-Table comporte 16 états (issus des combinaisons de discrétisation) et 10 actions. La récompense est fonction de la distance d :

$$r(d) = \begin{cases} -0.25 & \text{si } d > 1500, \\ -0.1 & \text{si } d \in]1200, 1500], \\ 0.2 & \text{si } d \in]900, 1200], \\ 0.5 & \text{si } d \in [700, 900], \\ 0.75 & \text{si } d \leq 700. \end{cases}$$

La **Figure 2** montre l'évolution du nombre d'étapes nécessaires pour terminer la course en fonction du nombre d'itérations d'apprentissage. On observe une diminution globale du nombre d'étapes, malgré quelques fluctuations, possiblement dues à l'initialisation nulle de la Q-Table et aux effets des récompenses négatives (phénomène d'*Optimistic Initial Values* selon [1]).

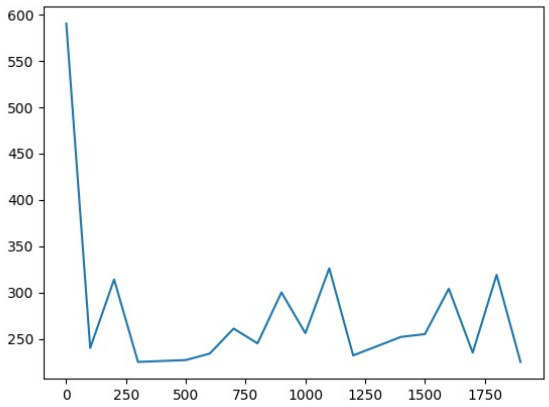

FIGURE 2 – Évolution du nombre d'étapes pour terminer la course en fonction des itérations d'apprentissage.

Enfin, pour évaluer la dépendance au simulateur, le temps d'entraînement sur 2000 tours (2% des épisodes du Tic Tac Toe) est de 9.53 secondes, soit 143.52% du temps nécessaire pour le Tic Tac Toe, soulignant l'influence du simulateur sur la qualité de l'apprentissage.

3.2 Méthode d'apprentissage par recherche arborescente (MCTS)

L'algorithme Monte Carlo Tree Search (MCTS) comme définit dans [2] est une méthode de recherche utilisée pour sélectionner la meilleure action en se basant sur des simulations répétées. Il se décompose en quatre phases principales :

1. **Sélection :** À partir de la racine (l'état courant du jeu), on parcourt l'arbre en choisissant à chaque étape le nœud qui maximise une formule d'exploration, typiquement l'UCT (Upper Confidence Bound applied to Trees).
2. **Expansion :** Si le nœud sélectionné n'est pas entièrement développé (c'est-à-dire qu'il existe des actions non encore explorées), un nouveau nœud correspondant à une action non explorée est ajouté à l'arbre.
3. **Simulation :** Une fois le nouveau nœud créé, une partie est simulée de manière aléatoire (ou semi-aléatoire) depuis cet état jusqu'à un état terminal, afin d'estimer le résultat de la partie.
4. **Rétropropagation :** Le résultat de la simulation est ensuite propagé en remontant l'arbre, mettant à jour pour chaque nœud traversé le nombre de visites et le nombre de victoires.

Au terme d'un nombre fixé de simulations, l'algorithme sélectionne l'action associée au meilleur enfant de la racine (souvent en fixant le paramètre d'exploration à zéro pour favoriser l'exploitation). Cette approche permet d'approximer une politique optimale sans avoir à explorer exhaustivement l'ensemble de l'espace d'états, ce qui est particulièrement utile pour des jeux à grande dimensionnalité.

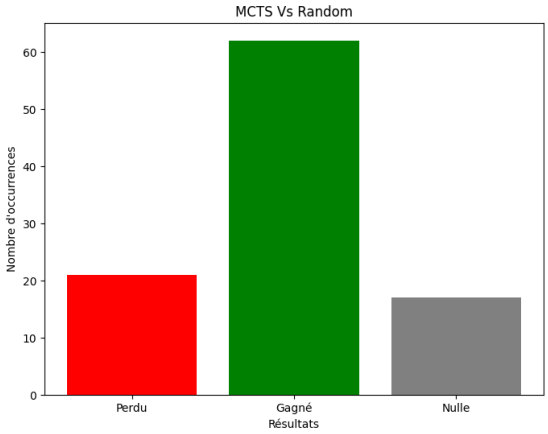

FIGURE 3 – MCTS Vs Random (17 nulles, 62 victoires, 21 défaites, WR = 62%, WR+Equity = 79%).

Les résultats obtenus en affrontant un adversaire aléatoire montrent que l'algorithme MCTS présente de bonnes performances, avec un taux de victoire de 62% et un taux combiné (victoires + nulles) de 79%.

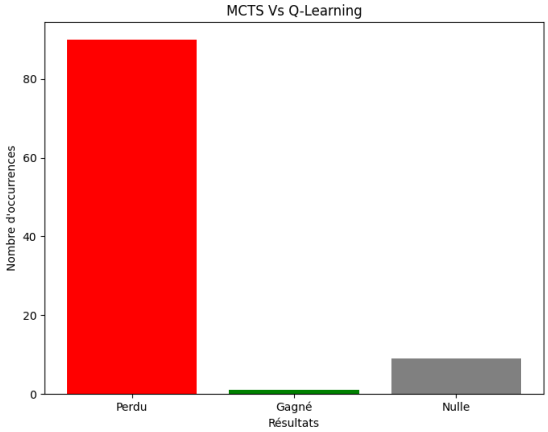

FIGURE 4 – MCTS Vs Q-Learning (9 nulles, 1 victoire, 90 défaites).

En revanche, face à un agent entraîné par Q-Learning, MCTS ne parvient qu'à obtenir 1 victoire sur 100 parties, ce qui souligne la supériorité de l'approche Q-Learning dans cet environnement.

4 Vers la deuxième phase du projet

Dans la suite du projet, notre objectif est de dépasser les limites du Q-Learning tabulaire, notamment l'explosion combinatoire du nombre d'états. En effet, pour des environnements complexes comme le Tic Tac Toe Ultimate, le nombre théorique d'états atteint 3^{81}, rendant une Q-Table explicite impraticable. Pour contourner ce problème, nous explorerons des approches de deep Q-learning. Le deep learning, ou apprentissage profond, consiste à utiliser des réseaux de neurones à plusieurs couches pour extraire automatiquement des représentations hiérarchiques des données. En l'appliquant au Q-Learning, nous pourrons approximer la fonction Q même dans des espaces d'états très volumineux et complexes, sans recourir à une table explicite.

Pour évaluer l'efficacité de notre approche de deep Q-learning, il est essentiel de définir des objectifs de convergence clairs et d'utiliser des métriques d'évaluation robustes. L'objectif principal est que la fonction Q approximée converge vers des valeurs stables, indiquant que l'agent a acquis une politique optimale maximisant la récompense cumulée. Pour ce faire, nous suivrons la réduction progressive de l'erreur de différence temporelle (TD Error) ainsi que la stabilisation des Q-values au fil des épisodes. Par ailleurs, nous évaluerons la performance de l'agent à travers des indicateurs spécifiques à chaque environnement, tels que le taux de victoires dans Tic Tac Toe ou la diminution du nombre d'étapes nécessaires pour terminer une course dans Mad Pod Racing. L'analyse des courbes de récompense cumulée et des temps de convergence nous permettra d'ajuster les hyperparamètres et d'assurer une généralisation efficace.

L'utilisation de deep Q-learning permettra de généraliser à partir d'exemples, de gérer efficacement des espaces continus et de grande dimension, et de répondre aux exigences de temps réel de telles plateformes. Nous poursuivrons également l'exploration d'algorithmes complémentaires (basés sur les policy gradients et les architectures actor-critic) pour concevoir un modèle capable de résoudre efficacement le problème du Tic Tac Toe Ultimate et d'optimiser les performances dans Mad Pod Racing. Ces travaux nous permettront d'analyser et de comparer les performances des différentes approches, en termes de convergence, de robustesse et de temps de calcul, et de proposer une solution robuste pour la création de bots optimisés.

Par ailleurs, nous chercherons à déterminer une méthode automatique pour identifier les seuils optimaux lors de la discrétisation des espaces d'états. Dans un premier temps, nous avons utilisé des critères simples tels que la distance et la différence d'angle ; toutefois, il sera nécessaire d'affiner ces seuils pour améliorer la qualité de l'apprentissage et réduire les fluctuations observées dans le simulateur Mad Pod Racing.

De plus, nous prévoyons de remplacer la simplification actuelle du calcul automatique de l'angle par une modélisation plus fine. Cette nouvelle approche intégrera d'autres niveaux de complexité, comme la présence d'obstacles, les collisions entre agents et d'autres interactions dynamiques, afin d'obtenir une représentation d'états plus précise et fidèle aux environnements réels.

Enfin, il est crucial de relever le défi de transférer nos agents sur la plateforme Coding Game, qui impose des contraintes de temps strictes pour chaque réponse. En effet, dans des jeux tels que Mad Pod Racing et Tic Tac Toe Ultimate, le délai alloué est de 1000 ms pour le premier coup, puis de 75 ms (MPR) et 100 ms (TTTU) pour les coups suivants. Ces contraintes rendent difficile l'exécution d'algorithmes complexes, et nécessitent une optimisation poussée du code ainsi que de l'architecture de nos agents pour garantir des prises de décision en temps réel sans dégrader la qualité de la stratégie adoptée. Ce passage vers Coding Game représente ainsi un enjeu majeur de notre projet, qui devra être abordé par des techniques d'optimisation et de transfert de modèles adaptées.

5 Conclusion

Nous avons implémenté un algorithme d'apprentissage par renforcement basé sur la valeur, le Q-Learning, ainsi que des simulateurs pour les jeux sélectionnés, à savoir le Tic-Tac-Toe Ultimate et le Mad Pod Racing. Nous avons analysé la performance de cet algorithme dans le cadre du Mad Pod Racing. Par ailleurs, nous avons également développé une méthode d'apprentissage par recherche arborescente, le Monte Carlo Tree Search (MCTS), pour explorer d'autres approches d'optimisation.

Références

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning : An Introduction*. MIT Press, 2nd edition, 2020.

[2] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. Monte carlo tree search : A review of recent modifications and applications. *Artificial Intelligence Review*, 56 :2497–2562, 2023.