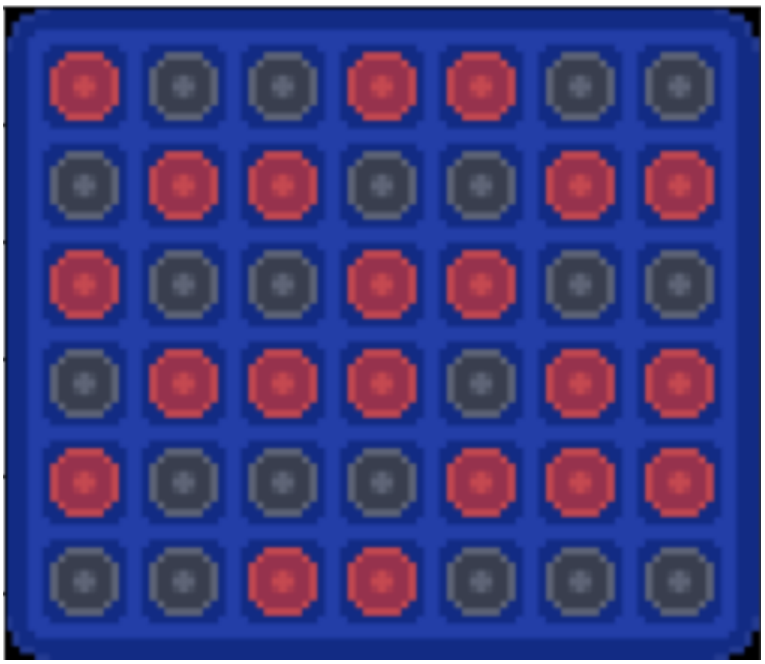
Techniques Monte Carlo Search Tree appliquées au cas du puissance 4

Antoine Gorceix

25 avril 2023



Contents

1 Introduction

L'IA de jeu est l'un des domaines de recherche les plus prometteurs notamment dans le domaine des jeux. Il s'agit d'apprendre à un agent à jouer à un jeu complexe. Récemment, il y a eu de nombreux travaux passionnants tels que AlphaGO de DeepMind et l'apprentissage de Hide and Seek par OpenAI. L'environnement de jeu offre de nombreux scénarios complexes pour tester plusieurs algorithmes qui peuvent ensuite être adaptés à des applications importantes dans la vie réelle, telles que la conduite autonome et la robotique. Dans ce rapport, vous trouverez une pour permettre à un agent d'apprendre à jouer à Puissance 4 selon deux **techniques Monte Carlo Search Tree**. Bien que simple pour les humains, l'espace d'action large et les différentes permutations et combinaisons d'états de plateau peuvent s'avérer difficiles pour un agent.
Présentons rapidement les concepts du jeu : Puissance 4 est un jeu où deux joueurs déposent chacun leur tour des pièces colorées dans une grille verticale. Le but de chaque joueur est de former une séquence de quatre disques en ligne avant son adversaire. C'est un jeu à **information parfaite**, ce qui signifie que chaque joueur est bien informé de tous les événements qui se sont produits auparavant. Alternativement, puissance 4 peut également être considéré comme un jeu à somme nulle, ce qui signifie qu'il n'y a ni victoire ni défaite mutuelle.

Problématique : Comment développer un agent qui sait jouer au puissance 4 avec des techniques de Monte Carlo Search Tree ?

Le code de ma partie du projet peut se trouver sur le repository github suivant : en cliquant ici
Le code du groupe peut se trouver sur le repository github suivant : en cliquant ici

2 Présentation des techniques utilisées

2.1 Monte Carlo Tree Search avec UCB

Il peut y avoir plusieurs algorithmes différents pour résoudre ce problème, au sein de mon groupe nous nous sommes ainsi répartis les tâches :

- Méthodes acteur-critique (AC, A2C, SAC, PPO) explorées par Karim Khaldi
- Algorithmes AlphaZero / MuZero explorés par Dylan Séchet
- Deep Q-Networks explorés par Leila Berrada
- Monte Carlo Tree Search que nous explorons dans ce rapport.

L'agent correspondant à la méthode MCTS UCB est implémentée dans le code sous le nom MCTS_UCB. L'algorithme de Monte Carlo Tree Search (MCTS) présente une certaine **flexibilité** pour des jeux avec un grand nombre de possibilités. Le principal obstacle pour les jeux impliquant un grand nombre d'actions (7 dans le cas de Connect4) est qu'ils nécessitent une recherche approfondie pour considérer les différentes permutations et combinaisons du plateau de jeu donné. MCTS tente de surmonter ce problème de manière efficace en **réduisant l'espace de recherche** tout en maintenant l'efficacité, comme expliqué ci-dessous.

Idée de la méthode: Monte Carlo Tree Search construit un arbre de recherche avec n noeuds, chaque noeud étant annoté avec le nombre de victoires et le nombre de visites. Initialement, l'arbre commence avec un seul noeud racine et effectue des itérations tant que les ressources ne sont pas épuisées.

MCTS se compose de quatre étapes principales :

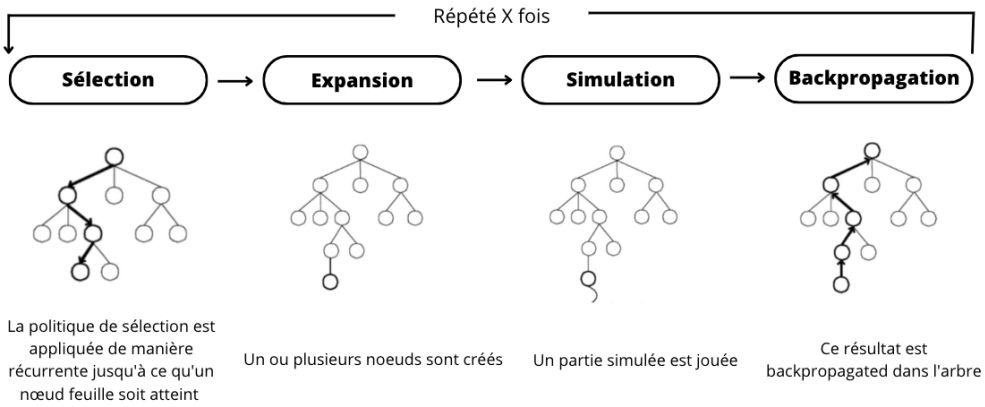


Figure 1: Les 4 étapes principales de Monte Carlo Tree Search

- **Configuration initiale** : Commencer avec un seul noeud racine (noeud parent) et assigner une grande valeur UCB aléatoire à chaque noeud non visité (noeud enfant).
- **Sélection** : Dans cette phase, l'agent commence par le noeud racine, sélectionne le noeud le plus urgent, applique les actions choisies et continue jusqu'à ce que l'état terminal soit atteint. Pour sélectionner le noeud le plus urgent, la borne supérieure de confiance des noeuds est utilisée. Le noeud avec le **UCB maximum** est utilisé comme prochain noeud. Le processus UCB aide à surmonter le dilemme **exploration-exploitation**. Aussi connu sous le nom de problème des bandits manchots où l'agent veut maximiser ses gains tout en jouant. La probabilité de sélectionner chaque noeud est : $v_i + C \times \sqrt{\frac{\log(N)}{n_i}}$
  Avec :
  - v_i : valeure estimée
  - C : paramètre que l'on peut faire varier
  - N : Nombre de visites du noeud parent
  - n_i : Nombre de visites

  Cette étape correspond dans le code à la méthode selection de la classe agent MCTS_UCB.
- **Expansion** : Lorsque UCB ne peut plus être appliqué pour trouver le prochain noeud, l'arbre de jeu est **étendu** pour inclure un enfant non exploré en ajoutant tous les noeuds possibles à partir du noeud feuille. Cette étape correspond dans le code à la méthode expansion de la classe agent MCTS_UCB.
- **Simulation** : Une fois étendu, l'algorithme sélectionne le noeud enfant soit au hasard, soit avec une politique jusqu'à ce qu'il atteigne l'étape finale du jeu. Cette étape correspond dans le code à la méthode simulation de la classe agent MCTS_UCB.
- **Backpropagation** : lorsque l'agent atteint l'état final du jeu avec un gagnant, tous les nœuds parcourus sont mis à jour. La visite et le score de victoire pour chaque nœud sont mis à jour. Cette étape correspond dans le code à la méthode backpropagation de la classe agent MCTS_UCB.

Les étapes ci-dessus sont répétées pour quelques itérations. Enfin, l'enfant du noeud racine avec le nombre le plus élevé de visites est sélectionné comme prochaine action, car plus le nombre de visites est élevé, plus le UCB est élevé. Cela se fait nottament dans le code grâce à la méthode get_action de la classe agent MCTS_UCB.

2.2 MCTS avec Rave

L'agent correspondant à la méthode MCTS Rave est implémentée dans le code sous le nom MCTS_rave.
Pour adapter la méthode Monte Carlo Search Tree (MCTS) au cas RAVE, il faut ajouter une étape supplémentaire avant la phase de sélection. Cette étape consiste à mettre à jour les valeurs RAVE pour chaque nœud visité lors de la simulation.

Les valeurs RAVE représentent la valeur d'un mouvement non exploré lors des simulations précédentes. En d'autres termes, si une simulation est effectuée et qu'un mouvement particulier n'est pas sélectionné, sa valeur RAVE est quand même mise à jour. Cela permet à la méthode MCTS de mieux tenir compte des mouvements qui ont été effectués lors des simulations précédentes, même s'ils n'ont pas été explorés lors de la phase de sélection.

La phase de sélection reste inchangée, où l'agent sélectionne le nœud le plus urgent en utilisant la borne supérieure de confiance des nœuds (UCB) appliquée aux valeurs Q et RAVE des nœuds enfants. La valeur Q représente la valeur de ce nœud basée sur les simulations précédentes, tandis que la valeur RAVE représente la valeur combinée des simulations précédentes et de la simulation actuelle.

Avant de passer à la phase de simulation, les valeurs RAVE des nœuds visités lors de la simulation précédente sont mises à jour en utilisant la valeur Q du nœud feuille visité. Cela signifie que si un mouvement particulier n'a pas été exploré lors de la phase de sélection, sa valeur RAVE sera mise à jour avec la valeur Q du nœud feuille visité lors de la simulation.

Pendant la simulation, les valeurs RAVE des nœuds visités sont également mises à jour en utilisant la valeur Q du nœud feuille visité. Ainsi, la valeur RAVE continue d'être mise à jour même pendant la phase de simulation, ce qui permet de mieux tenir compte des mouvements qui ont été effectués lors des simulations précédentes.

Lorsque l'agent atteint l'état final du jeu avec un gagnant, tous les nœuds parcourus sont mis à jour lors de la phase de backpropagation. La visite et le score de victoire pour chaque nœud sont mis à jour en utilisant la valeur Q et RAVE mise à jour. Cela permet à l'algorithme MCTS de mettre à jour les valeurs Q et RAVE pour chaque nœud visité, en prenant en compte à la fois les mouvements explorés lors de la phase de sélection et les mouvements non explorés mis à jour lors de la phase de simulation.

Pour résumer MCTS :

- Chaque itération commence à la racine
- Suit la politique de l'arbre pour atteindre un noeud feuille
- Le noeud N est ajouté
- Effectuez un lancement aléatoire
- La valeur est rétropropagée dans l'arbre

3 Expériences et résultats obtenus

Plusieurs expériences ont été réalisées, tout d'abord nous avons fait joué nos deux modèles contre l'agent Random. Nous obtenons plutôt de bons résultats : 95% de victoires pour la méthode Monte Carlo Tree Search Rave et 87% pour la méthode MCTS UCB :

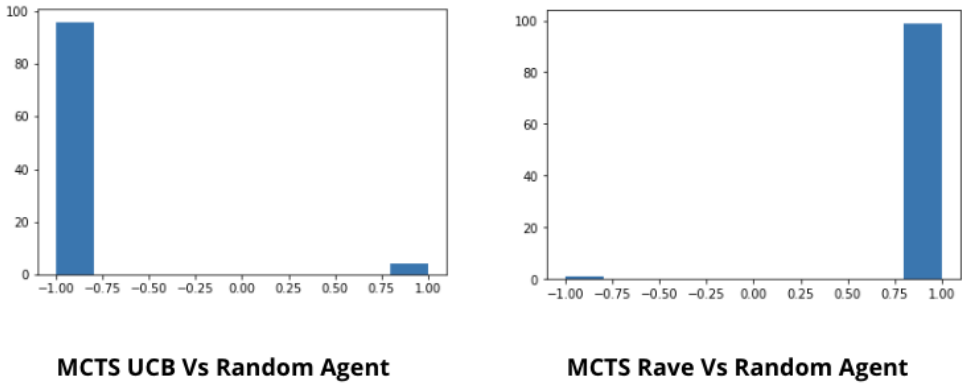


Figure 2: Comparaison des performances face au Random Agent

La méthode MCTS rave semble donc un peu plus performante que la méthode MCTS UCB. Les résultats obtenus ci-dessus, sont obtenus dans le cas où l'agent MCTS commence. L'intuition développée en faisant jouer deux agents Random l'un contre l'autre se confirme : lorsque l'agent random commence, les performances des agents MCTS sont légèrement moins bonnes : 93% pour rave et 89% pour UCB.

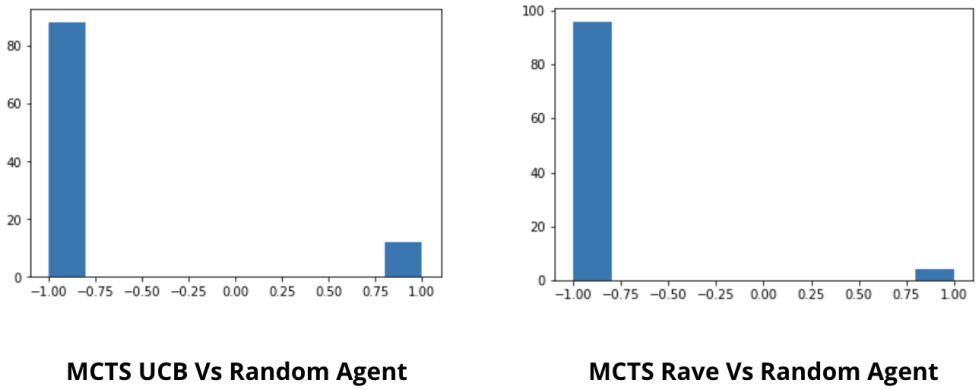


Figure 3: Comparaison des performances face au Random Agent en effectuant le premier coup

Une autre expérience a été menée : une confrontation entre les deux agents MCTS rave et MCTS UCB. Ces deux méthodes étants très chronophages, uniquement 12 confrontations ont été effectuées.
Le résultat est présenté dans la figure ci-dessous :

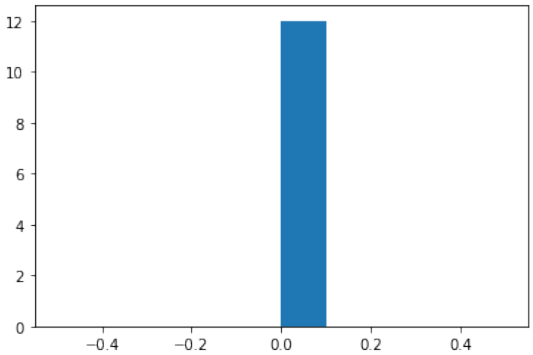


Figure 4: Confrontation MCTS Rave Vs MCTS UCB

Un exemple de plateau obtenu après égalité est donné en illustration au début du rapport. Pour douze confrontation,

on obtient douze égalité. Il est donc intéressant de voir ce que cela donne en laissant intact les hyperparamètres de l'un et en faisant varier ceux de l'autre. Considérons l'hyperparamètre timeout, il détermine dans la durée limite de la boucle while de la méthode get_action. Il conditionne ainsi la durée de l'exploration et donc la profondeur des arbres explorés. Il était fixé à 2 dans nos précédentes expériences pour les deux agents. Augmenter la valeur de cet hyperparamètre augmente grandement le temps de calcule, c'est la raison pour laquelle nous l'avions choisi faible. Nous le fixons à 5 pour MCTS UCB et le laissons à 2 pour MCTS rave. Nous obtenons des résultats différents : sur 12 confrontations, 9 ont été gagnées par MCTS UCB et 3 par MCTS Rave. On en conclut qu'en augmentant la durée d'éxploration de nos agents, on améliore leur performances.

4 Conclusion

Dans le cadre de notre étude, nous avons appliqué les méthodes Monte Carlo Tree Search Rave et UCB pour traiter le cas du puissance 4. Ces méthodes ce sont avérées très efficaces puisque leur performance tournent en moyenne autour de 90% face à un agent random. Néanmoins, le cas auquel nous avons appliqué ces deux méthodes n'étaient pas très complexes. En effet, les méthodes de recherche arborescente telles que MCTS rave et UCB peuvent être efficaces dans des environnements plus complexes comme les échecs, le backgammon ou Starcraft car elles peuvent explorer efficacement des espaces d'états plus vastes en utilisant des heuristiques.
Si nous avions eu plus de temps ou de puissance de calcul est disponible nous aurions pu explorer des améliorations plus avancées de MCTS telles que les variantes de MCTS basées sur des réseaux de neurones, telles que AlphaGo ou AlphaZero. Ces approches utilisent des réseaux de neurones pour apprendre des politiques et des fonctions de valeur pour guider l'exploration de l'arbre.