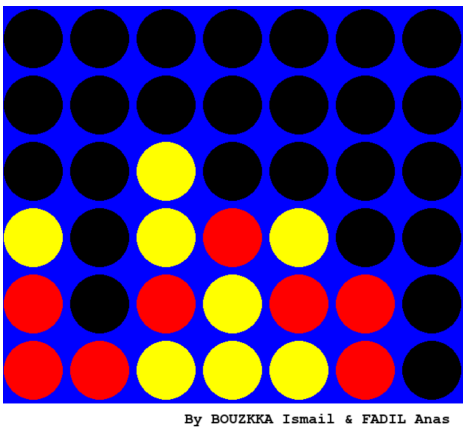
IA pour Puissance 4 utilisant Monte Carlo Tree Search (MCTS) avec UCB, RAVE, PUCT et NRPA

BOUZKKA Ismail & FADIL Anas

31 July 2024



1 Introduction

Ce projet implémente le jeu Puissance 4 utilisant la recherche d'arbre Monte Carlo (MCTS) améliorée avec les bornes supérieures de confiance pour les arbres (UCB), l'estimation de valeur d'action rapide (RAVE) ,prédicteur + Limites de Confiance Supérieures pour les Arbres (PUCT) et l'adaptation de la politique de déploiement imbriquée (NRPA). L'algo prend des décisions en simulant des résultats de jeu et en utilisant l'analyse statistique pour choisir le meilleur coup. Dans la suite de ce rapport, on explique les concepts utilisés ainsi que l'implémentation du code.

2 Concepts Utilisés

2.1 Jeu Puissance 4

Puissance 4 est un jeu de plateau pour deux joueurs où les joueurs déposent à tour de rôle des disques colorés dans une grille verticale. L'objectif est de connecter quatre de ses disques en ligne, soit horizontalement, verticalement ou en diagonale, avant l'adversaire.

2.2 Monte Carlo Tree Search (MCTS)

MCTS est un algorithme de recherche utilisé pour les processus de prise de décision, en particulier dans les jeux. Il comporte quatre étapes principales :

1. **Sélection** : À partir de la racine, sélectionner un nœud enfant jusqu'à atteindre un nœud feuille.
2. **Expansion** : Étendre le nœud feuille en ajoutant un ou plusieurs nœuds enfants.
3. **Simulation** : Simuler un déroulement aléatoire depuis le nouveau nœud jusqu'à un état terminal.
4. **Rétropropagation** : Propager le résultat de la simulation à travers l'arbre pour mettre à jour les nœuds.

2.3 Upper Confidence Bounds for Trees (UCB)

UCB est une stratégie utilisée pour équilibrer exploration et exploitation lors de la phase de sélection de MCTS. Elle calcule la valeur de chaque nœud en utilisant la formule :

$$\text{UCB}(n) = \frac{w_i}{n_i} + c\sqrt{\frac{\ln t}{n_i}}$$

où :

- w_i : Nombre de victoires après le i-ème coup.
- n_i : Nombre de simulations après le i-ème coup.
- c : Paramètre d'exploration (théoriquement égal à $\sqrt{2}$).
- t : Nombre total de simulations pour le nœud parent.

Cette formule permet de choisir le premier mouvement à la racine selon l'UCB avant chaque simulation.

2.4 Rapid Action Value Estimation (RAVE)

RAVE est une variante de MCTS qui combine les valeurs d'action des simulations précédentes pour accélérer la convergence de l'algorithme. La formule de RAVE est donnée par :

$$Q_*(s,a) = (1 - \beta(s,a))Q(s,a) + \beta(s,a)\tilde{Q}(s,a)$$

où :

- $Q(s,a)$: Valeur de l'action a dans l'état s selon MCTS.
- $\tilde{Q}(s,a)$: Valeur de l'action a dans l'état s selon toutes les simulations (AMAF).
- $\beta(s,a)$: Poids donné à $\tilde{Q}(s,a)$, calculé par :

$$\beta = \frac{\tilde{n}}{n + \tilde{n} + n\tilde{n}\bar{b}^2/\mu_*(1-\mu_*)}$$

La méthode RAVE utilise également les étapes standard de MCTS, mais elle améliore la sélection des coups en intégrant les informations globales des simulations, ce qui permet une convergence plus rapide vers des décisions optimales.

2.5 PUCT

PUCT est une variante de MCTS qui intègre les prédictions de valeur et de politique fournies par le modèle prédictif pour guider l'exploration de l'arbre de recherche. Utilisée notamment dans AlphaGo et AlphaZero, PUCT combine les probabilités de mouvement fournies par le modèle prédictif avec la formule d'UCB pour équilibrer exploration et exploitation.

$$U(s,a) = c_{puct} \cdot P(s,a) \cdot \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)}$$

où :

- $P(s,a)$: Probabilité prédite pour le mouvement a dans l'état s.
- $N(s,a)$: Nombre de fois que le mouvement a a été sélectionné dans l'état s.
- c_{puct} : Paramètre de contrôle de l'exploration.

2.6 Nested Rollout Policy Adaptation (NRPA)

NRPA est une méthode qui adapte dynamiquement la politique de déploiement pendant la recherche pour améliorer l'efficacité et les résultats. Contrairement aux méthodes de MCTS traditionnelles qui utilisent des politiques statiques, NRPA utilise le gradient pour ajuster la politique de déploiement à chaque niveau de recherche imbriquée.

L'algorithme suit les étapes suivantes :

1. **Déploiement (Rollout)** : À chaque niveau, une série de déploiements aléatoires est effectuée en utilisant la politique actuelle. Les mouvements sont sélectionnés en fonction d'une distribution de probabilité basée sur la politique actuelle, représentée par un vecteur de poids où chaque mouvement possible a un poids associé.

2. **Adaptation de la Politique (Policy Adaptation)** : La politique est adaptée en augmentant les poids des mouvements présents dans les meilleures séquences trouvées. Cette adaptation est réalisée par une méthode de montée de gradient, qui ajuste les poids pour augmenter les probabilités des mouvements réussis tout en normalisant les poids pour maintenir une distribution de probabilité valide.

3. **Recherche Imbriquée (Nested Search)** : La recherche se poursuit de manière récursive en appelant le niveau inférieur de l'algorithme jusqu'à atteindre le niveau de base. Les résultats des niveaux inférieurs sont utilisés pour guider et adapter la recherche aux niveaux supérieurs.

4. **Mise à Jour (Update)** : À chaque niveau supérieur, plusieurs itérations de la recherche sont effectuées. Chaque itération utilise la politique actuelle pour exécuter des déploiements à un niveau inférieur. Les résultats des déploiements sont utilisés pour identifier les meilleures séquences de mouvements, et la politique de déploiement est ensuite adaptée pour favoriser ces mouvements.

3 Implémentation du Code

3.1 Bibliothèques et Initialisation

Nous utilisons les bibliothèques suivantes :

- `numpy` pour la gestion du plateau de jeu.
- `pygame` pour la représentation graphique et l'interaction utilisateur.
- `math` et `random` pour les calculs et les simulations aléatoires.

3.2 Couleurs et Dimensions

Nous définissons les couleurs et les dimensions pour le plateau de jeu et les éléments de l'interface utilisateur :

3.3 Classe `ConnectFourBoard`

Cette classe encapsule la logique du jeu et la gestion du plateau :

- `create_board` : Initialise un plateau de jeu vide.
- `drop_piece` : Insère une pièce dans une colonne spécifique.

- `is_valid_location` : Vérifie si une colonne peut recevoir une pièce.
- `get_next_open_row` : Trouve la prochaine ligne disponible dans une colonne.
- `print_board` : Affiche l'état actuel du plateau.
- `winning_move` : Vérifie si un coup est gagnant (si le joueur spécifié a une combinaison gagnante).
- `legal_moves` : Retourne une liste des mouvements légaux (une liste de colonnes valides pour un coup).
- `terminal` : Vérifie si le jeu est terminé.
- `score` : Calcule le score de la partie.
- `play` : Joue un coup sur le plateau.
- `playout` : Simule une partie aléatoire depuis l'état actuel jusqu'à un état terminal.
- `draw_board` : Dessine l'état actuel du plateau en utilisant Pygame.

3.4 Implémentation de `ucb`

`ucb` : Utilise l'algorithme UCB pour sélectionner les mouvements. Voici comment il est implémenté :

1. **Initialisation** : Pour chaque mouvement légal, initialiser les scores et les visites.
2. **Sélection** : Pour chaque simulation, sélectionner le mouvement avec la plus haute valeur UCB.
3. **Simulation** : Effectuer une simulation (playout) pour le mouvement sélectionné.
4. **Mise à jour** : Mettre à jour les scores et les visites en fonction du résultat de la simulation.

3.5 Implémentation de `rave`

`rave` : Utilise l'algorithme RAVE pour sélectionner les mouvements. Voici comment il est implémenté :

1. Calcul des valeurs de $Q(s, a)$ et $\tilde{Q}(s, a)$ pour chaque action.
2. Calcul du poids $\beta(s, a)$ en utilisant les formules données.
3. Mise à jour des scores et des visites après chaque simulation.
4. Sélection de l'action avec la meilleure valeur $Q_*(s, a)$.

3.6 Implémentation de `puct`

`puct` : Utilise l'algorithme PUCT pour sélectionner les mouvements. Voici comment il est implémenté :

1. **Initialisation** : Initialiser les scores et les visites pour chaque mouvement légal.
2. **Calcul des valeurs** : Calculer les valeurs de $U(s, a)$ pour chaque mouvement en combinant les probabilités prédictives et les valeurs UCB.
3. **Sélection** : Pour chaque simulation, sélectionner le mouvement avec la plus haute valeur $U(s, a)$.
4. **Simulation** : Effectuer une simulation (playout) pour le mouvement sélectionné.
5. **Mise à jour** : Mettre à jour les scores et les visites en fonction du résultat de la simulation.

3.7 Implémentation de `nrpa`

`nrpa` : Utilise l'algorithme NRPA pour sélectionner les mouvements. Voici comment il est implémenté :

1. **Adaptation de la Politique** : Utilise l'ascension de gradient pour ajuster la politique basée sur les séquences de déploiement réussies.
2. **Déploiement** : Effectue des déploiements aléatoires selon la politique actuelle et enregistre les séquences.
3. **Recursion** : Pour chaque niveau de recherche, appelle récursivement le niveau inférieur avec la politique mise à jour.
4. **Mise à Jour** : Ajuste la politique basée sur les meilleures séquences trouvées.

3.8 Modes de Jeu

Le programme propose plusieurs modes de jeu :

- Jeu multijoueur : Deux joueurs humains s'affrontent.
- Jeu contre l'IA : Le joueur humain joue contre une IA utilisant l'un des algorithmes cités (UCB, RAVE, PUCT, NRPA).
- IA contre IA : deux IA utilisant l'un des algorithmes cités jouent l'une contre l'autre.

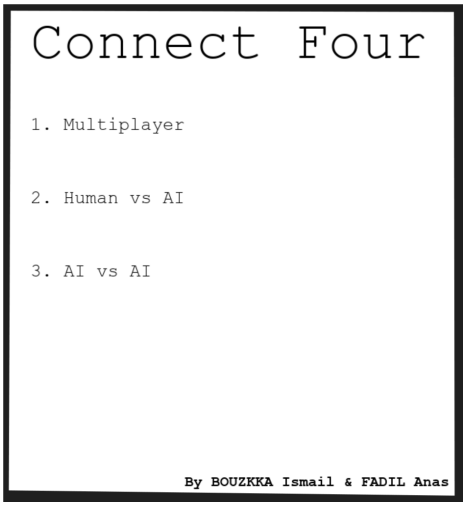


Figure 1: Menu du jeu Puissance 4

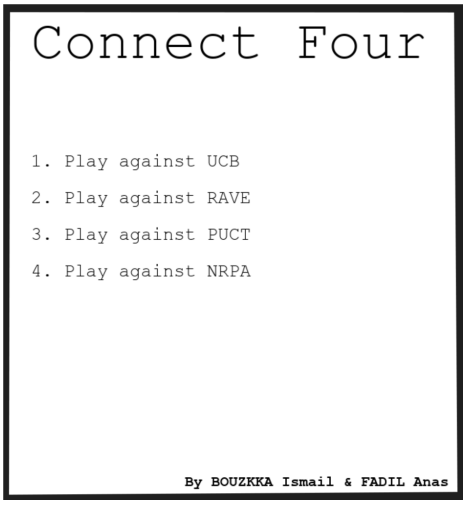


Figure 2: Choix Human vs Algo

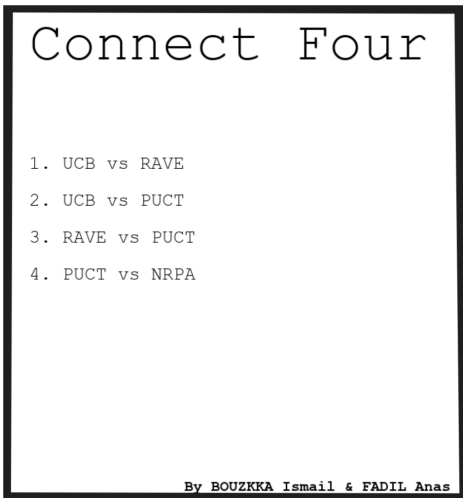


Figure 3: Choix Algo vs Algo

Algorithme	**Forces**	**Faiblesses**
UCB	Simplicité, bonne performance générale	Convergence lente, inefficace pour actions redondantes
RAVE	Convergence rapide, utilise des infos globales	Complexité accrue, valeurs AMAF non représentatives
PUCT	Utilise prédictions guidées par modèles	Nécessite modèle prédictif, complexité élevée
NRPA	Politique adaptative, amélioration continue	Complexité accrue, gestion des poids

Table 1: Comparaison des Algorithmes MCTS

4 Conclusion

Cette implémentation démontre l'utilisation de la recherche d'arbre Monte Carlo (MCTS) améliorée avec les différentes méthodes citées pour créer une IA qui joue au Puissance 4. En simulant des résultats de jeu et en analysant des données statistiques, l'algo prend des décisions informées pour choisir le meilleur coup.