Reversi: Optimisation algorithmique et mémoire

LAICHE GHODBANE BAADACHE

2 juin 2025

Plan de la présentation

Développement incrémental de notre IA

Progression méthodique

- **Étapes 1-2** : Implémentation du jeu et IA aléatoire
- **Étape 3** : Arbre de jeu et Minimax (évaluation par différence de pions)
- **Étape 4** : Arbre de jeu (profondeur variable) et détection des ensembles alignés (2 et 3+)
- **Étape 5** : Alpha-beta et une évaluation des positions stratégiques

Innovations majeures présentées

- **Optimisation mémoire** : réduction de 98% de l'empreinte mémoire
- **NegaScout avec tri des coups** : amélioration d'Alpha-Beta
- **Détection d'endgame** : stratégie spécialisée en fin de partie

Optimisation mémoire : problématique et approche

Problématique

- Explosion exponentielle de la mémoire avec la profondeur
- Limites pratiques pour explorer au-delà de la profondeur 6
- Construction d'arbres complets trop coûteuse en mémoire

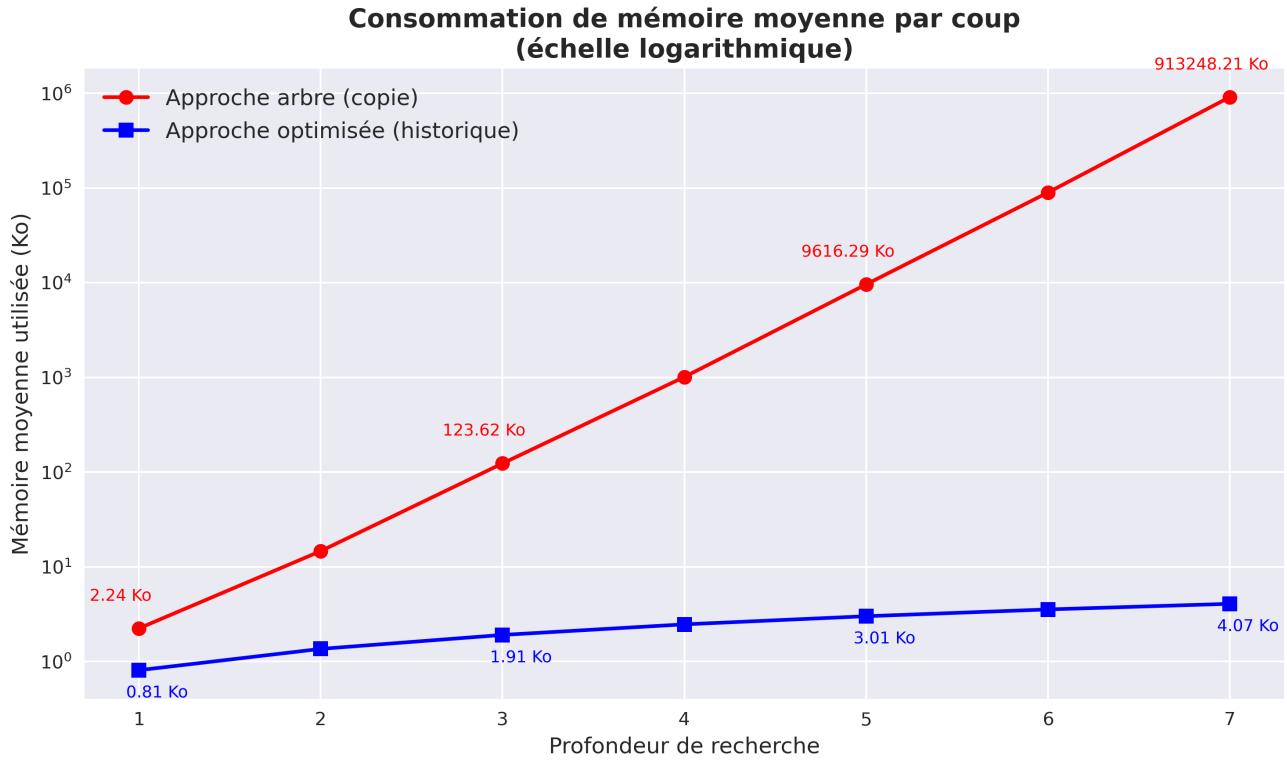
Notre approche innovante

- Un seul `Plateau` au lieu d'un arbre complet
- Structure `HistoriqueCoup` pour enregistrer les modifications
- Application d'un coup puis annulation systématique
- Compteurs précis pour suivre la mémoire utilisée

Protocole expérimental

- Comparaison de deux approches sur environ 30 plateaux simulant une partie
- Mesure de la mémoire utilisée pour différentes profondeurs (1 à 7)
- Analyse automatisée des résultats (script Python)

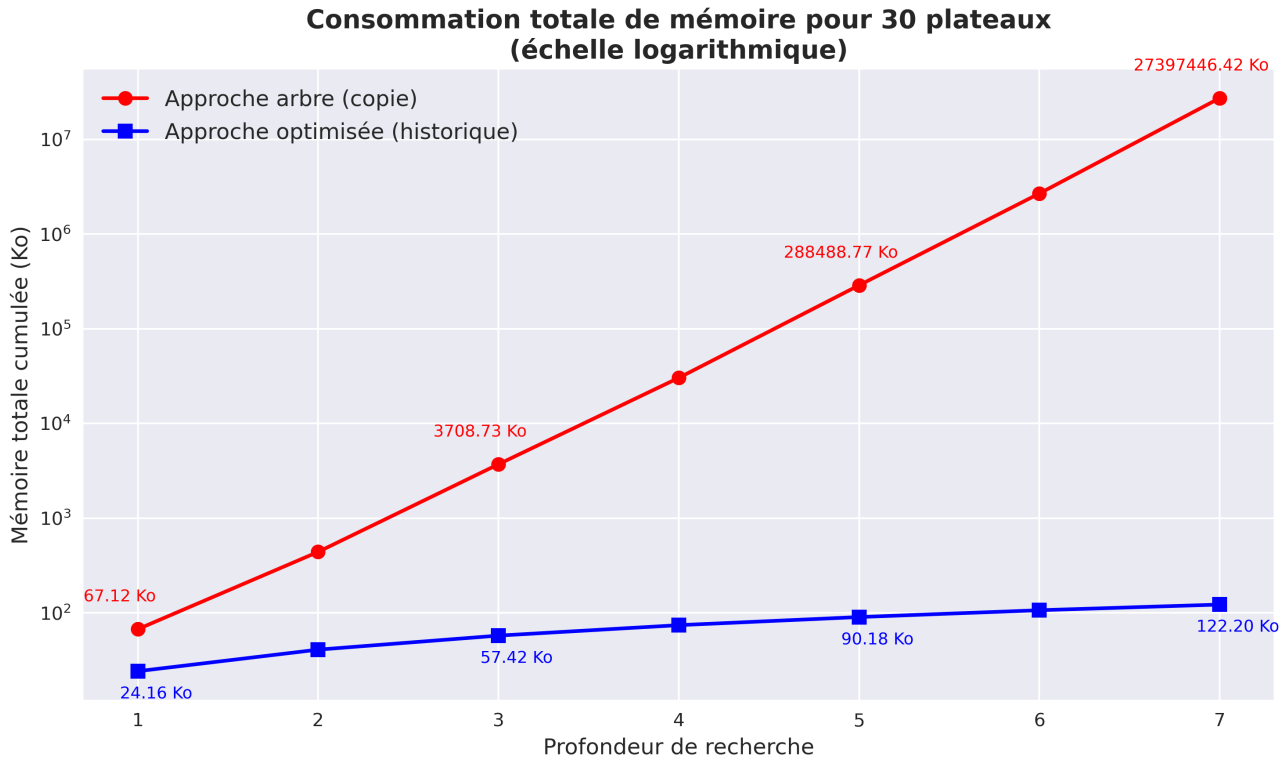
Optimisation mémoire : consommation moyenne par coup



Analyse des résultats

- **À profondeur 3** : 123,62 Ko vs 1,91 Ko = réduction de **98,5%**
- **À profondeur 5** : 9616,29 Ko vs 3,01 Ko = réduction de **99,97%**
- **À profondeur 7** : 913248,21 Ko vs 4,07 Ko = réduction de **99,99%**
- La différence s'accentue exponentiellement avec la profondeur

Optimisation mémoire : impact sur la partie complète



Implications pratiques

- **Profondeur désormais accessible** : jusqu'à 12 niveaux (contre 5-6 maximum avant)
- **Performance globale** : même avec 30 coups dans une partie, la mémoire reste gérable
- **À profondeur 7** : consommation réduite de 27 Mo à seulement 122 Ko
- Permet l'exploration d'arbres beaucoup plus profonds, impossible autrement

NegaScout : principe et variantes testées

Principe des fenêtres nulles

- Premier coup : fenêtre complète `[alpha, beta]`
- Coups suivants : fenêtre nulle `[alpha, alpha+1]` ou `[beta-1, beta]`
- Réévaluation avec fenêtre complète uniquement si le coup est prometteur

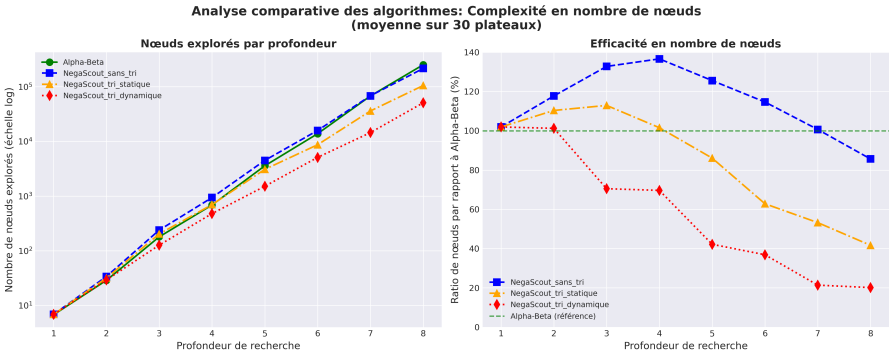
Variantes implémentées et testées

- **Alpha-Beta** : référence, algorithme classique sans fenêtres nulles
- **NegaScout sans tri** : coups explorés dans l'ordre initial
- **NegaScout avec tri statique** : coups triés selon des heuristiques avant exploration
- **NegaScout avec tri dynamique** : tri basé sur l'évaluation complète de chaque coup

Protocole de test

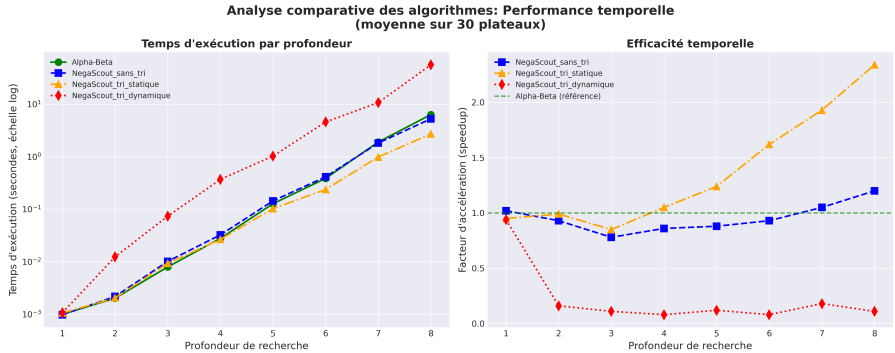
- Test sur 30 plateaux représentatifs
- Mesures : nombre de nœuds explorés et temps d'exécution
- Profondeurs testées : 1 à 8

NegaScout : analyse comparative des performances



Analyse du nombre de nœuds explorés

- **Tri dynamique** : plus efficace (jusqu'à 80% de réduction)
- **Tri statique** : bon compromis (40-60% de réduction à grande profondeur)
- **NegaScout sans tri** : inefficace à faible profondeur, mais s'améliore



Analyse du temps d'exécution

- **Tri statique** : nettement plus rapide (jusqu'à 2,3× d'accélération)
- **Tri dynamique** : malgré moins de nœuds, trop coûteux en temps
- **NegaScout sans tri** : performances comparables à Alpha-Beta

Conclusion

- Le tri statique offre le meilleur compromis performance/temps et a été retenu
- À grande profondeur, il réduit de 60% les nœuds explorés et accélère de 230% le temps

Détection d'endgame et évaluation adaptative

Détection de l'endgame

- Critères précis : cases vides ≤ 14, mobilité ≤ 8, coins ≥ 3
- Augmentation auto. de profondeur ($+6$)
- Recherche exhaustive pour score optimal

Critères stratégiques

- **Mobilité** : coups légaux, frontière
- **Stabilité** : pions inretournables
- **Parité** : avantage cases vides pair/impair
- **Disk-Square Tables** : valeurs positionnelles

Pondération par phase

- **Début** : mobilité ($\times 5$), positions ($\times 2$)
- **Milieu** : mobilité ($\times 4$), stabilité ($\times 3$), ensembles ($\times 3$)
- **Fin** : stabilité ($\times 6$), parité ($\times 3$)

Conclusion et résultats obtenus

Améliorations quantifiables

- **Mémoire** : réduction de 98% à profondeur 3, jusqu'à 99,99% à profondeur 7
- **Performance** : NegaScout avec tri statique explore 60% moins de nœuds et accélère le temps de 230%
- **Endgame** : détection précise permettant d'augmenter la profondeur de 6 niveaux

Compétences développées

- Optimisation algorithmique et gestion fine de la mémoire
- Analyse comparative et prise de décision basée sur les données
- Implémentation d'heuristiques avancées pour l'IA de jeux
- Développement incrémental avec validation expérimentale

Merci de votre attention