POLYTECH LYON

Lyon 1

Intelligence Artificielle Partie 3

M. Stéphane Bonnevay

4A Informatique

Rapport IA

Rédigé par
Amaury GALICHET
&
Cerine KERARMA

Rendu le 15 juin 2025

Sommaire

*le lien de notre repository git est en bibliographie

Introduction **1**
I. Fonctions d'évaluation **2**
II. Tests **2**
1. Évaluation AlphaBeta et Minimax 2
AlphaBeta vs. AlphaBeta 3
Minimax vs Minimax 5
2. Comparaison de AlphaBeta avec Minimax 6
Interprétation des résultats 7
3. Évaluation MCTS 7
Interprétation des résultats 9
4. Comparaison de Alphabeta et Minimax avec MCTS 9
Interprétation des résultats 10
CONCLUSION **10**

Introduction

L'intelligence artificielle (IA) joue un rôle majeur dans le domaine des jeux, en particulier pour les jeux à deux joueurs à information complète, comme les échecs, le Go ou le Puissance 4. Ces jeux offrent un cadre idéal pour tester et comparer différentes stratégies algorithmiques, car ils combinent des règles bien définies avec une complexité variable selon la profondeur d'analyse.

Les algorithmes classiques, tels que Minimax et son optimisation Alpha-Beta, permettent à une IA d'évaluer les coups possibles en explorant un arbre de jeu de manière systématique. Ces méthodes reposent sur des fonctions d'évaluation heuristiques pour estimer l'avantage d'une position. Plus récemment, des approches probabilistes comme Monte Carlo Tree Search (MCTS) ont émergé, offrant une alternative efficace en simulant des parties aléatoires pour guider la prise de décision.

Dans ce projet, nous avons choisi d'implémenter le jeu du Puissance 4, un jeu emblématique à information complète, pour comparer les performances des algorithmes Minimax, Alpha-Beta et MCTS. Notre objectif est d'analyser leurs comportements en termes de complexité algorithmique (nombre de nœuds explorés, temps de réponse), qualité de jeu (capacité à prendre des décisions optimales) et paramétrisation (influence des heuristiques et des simulations sur MCTS).

Nous détaillerons d'abord la conception du jeu, l'implémentation des différentes IA puis nous effectuerons une série de tests pour évaluer et comparer ces algorithmes. Enfin, nous analyserons les résultats afin de déterminer les avantages et limites de chaque approche.

I. Fonctions d’évaluation

Nous avons implémenté trois fonctions d’évaluation distinctes, chacune axée sur une stratégie différente. Ces fonctions permettent de mesurer l’efficacité des IA en fonction de leur approche et seront comparées pour chaque algorithme.

Fonction 1: Évaluation Standard (Basée sur le Cours)
Cette fonction suit une approche équilibrée, attribuant des poids exponentiels aux alignements de jetons pour refléter leur importance stratégique :

- 4 pions alignés (victoire) : 1000 points
- 3 pions + 1 case vide : 50 points (menace immédiate)
- 2 pions + 2 cases vides : 5 points (potentiel futur)
- 1 pion + 3 cases vides : 1 point (influence minimale)

Fonction 2: Évaluation Offensive
Cette version est conçue pour maximiser l’agressivité de l’IA en accordant une grande importance aux alignements de 3 jetons :

- 4 pions alignés (victoire) : 10 000 points (priorité absolue)
- 3 pions + 1 case vide : 200 points (très forte incitation à compléter un alignement gagnant)
- 2 pions + 2 cases vides : 10 points (potentiel secondaire)
- 1 pion + 3 cases vides : 1 point (impact minimal)

Fonction 3: Évaluation Défensive
Cette approche minimise les risques en ignorant les alignements isolés (1 pion) et en réduisant l'impact des petits alignements :

- 4 pions alignés (victoire) : 1000 points
- 3 pions + 1 case vide : 30 points (seulement si le blocage est urgent)
- 2 pions + 2 cases vides : 2 points (très faible poids car peu dangereux à court terme)
- 1 pion + 3 cases vides : 0 point (ignoré)

II. Tests

1. Évaluation AlphaBeta et Minimax

Nous avons implémenté trois algorithmes d'IA pour le Puissance 4: Minimax (avec élagage alpha-bêta), MCTS (Monte Carlo Tree Search) et Alpha-Beta (optimisation de Minimax). Il y a une configuration Joueur contre Joueur, Joueur contre IA ou IA contre IA (avec choix d'algorithme pour chaque IA).

Pour les tests, dans un premier temps nous avons choisi de faire s'affronter les IA Alpha-Beta et Minimax entre elles, c'est-à-dire :

- Alpha-Beta vs Alpha-Beta
- Minimax vs Minimax

Les profondeurs de recherche testées sont les suivantes :

- De 1 à 8 pour Alpha-Beta
- De 1 à 6 pour Minimax

Nous avons été limités par la capacité mémoire même après l'avoir augmentée. Ces profondeurs représentent donc le maximum atteignable pour chacun des algorithmes dans notre environnement d'exécution et nous nous sommes contentés de ces valeurs pour les tests.

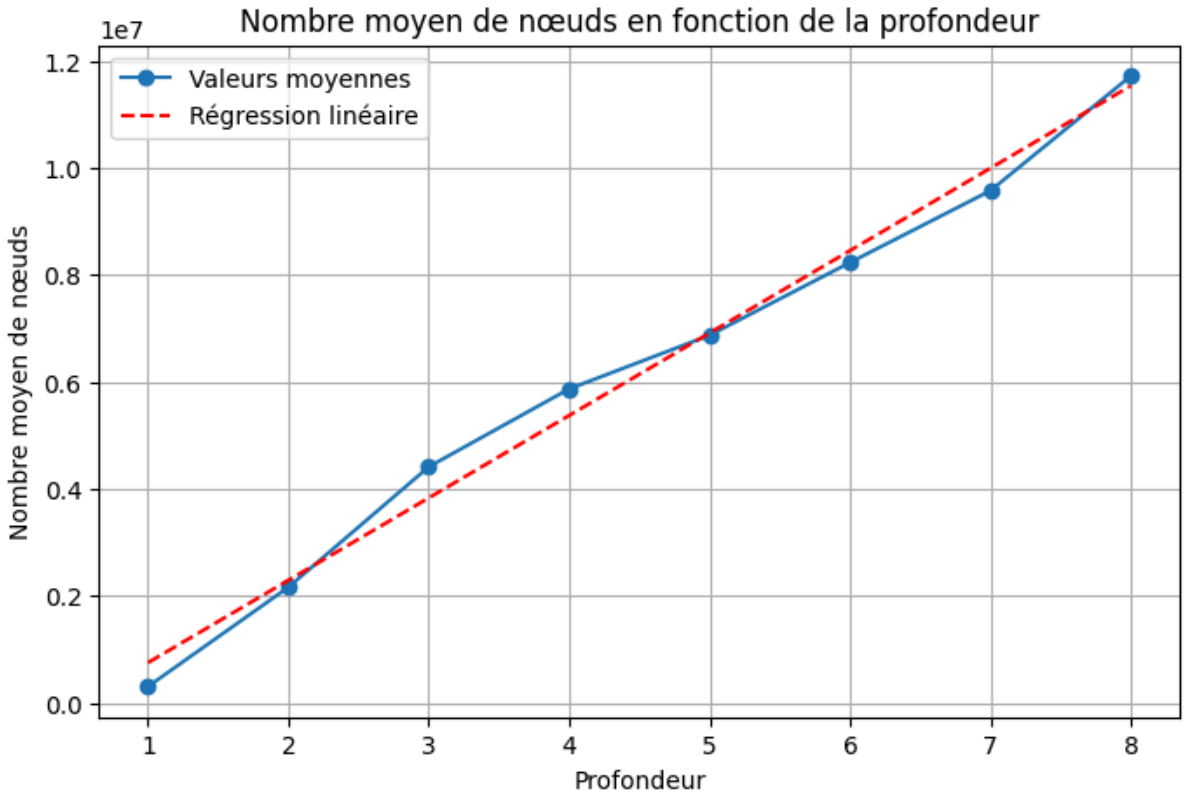
Les résultats suivants correspondent à la fonction d'évaluation standard

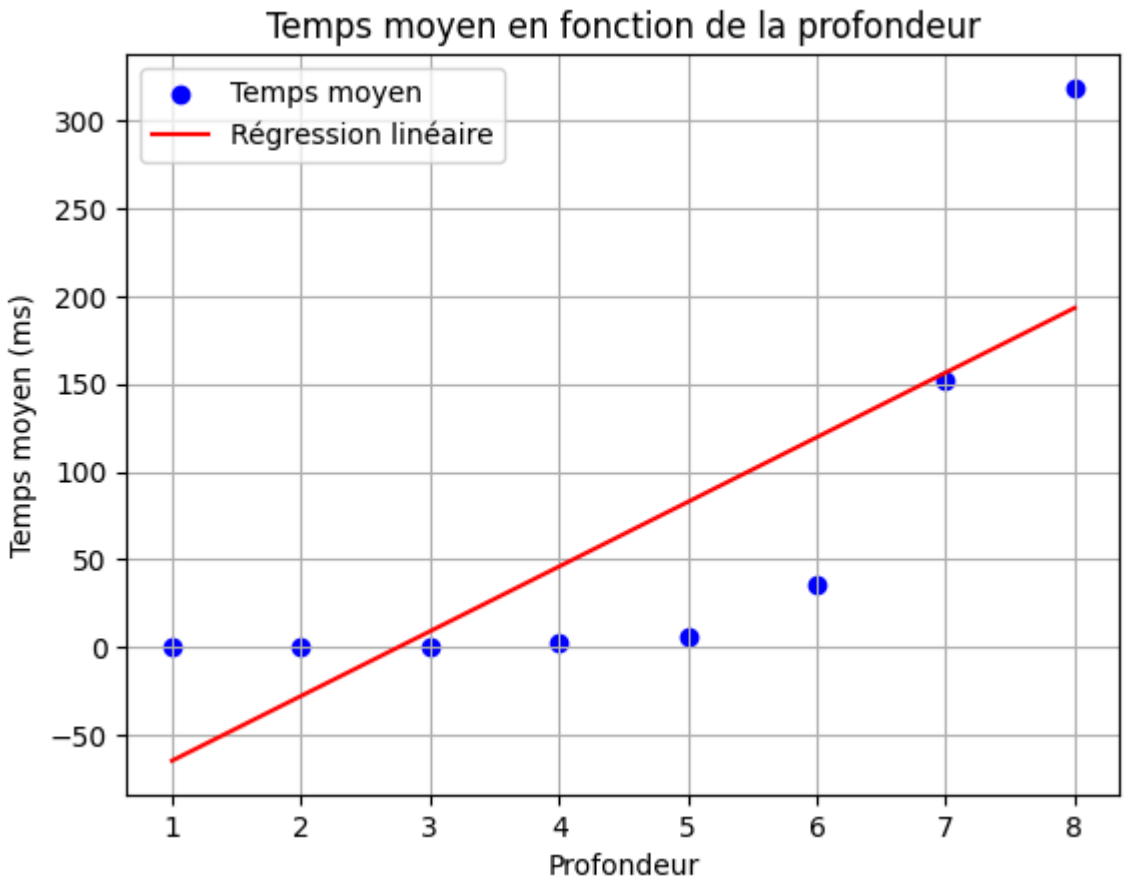
AlphaBeta vs. AlphaBeta

Profondeur	Nombre de noeuds moyen	Temps moyen (ms)	Victoires	Vainqueur dominant
1	3M*	0.350	22%	souvent perdant
2	2M	0.075	88%	souvent gagnant
3	4M	0.333	44%	souvent perdant
4	5M	1.328	44%	souvent perdant
5	6M	4.120	44%	souvent perdant
6	8M	20.902	44%	souvent perdant
7	9M	89.360	44%	souvent perdant
8	1B	202.298	66%	souvent gagnant

**M pour millions et B pour milliards*

Tableau 1 : Résultats de l'analyse de l'algorithme Alpha-Beta contre lui-même selon différentes profondeurs

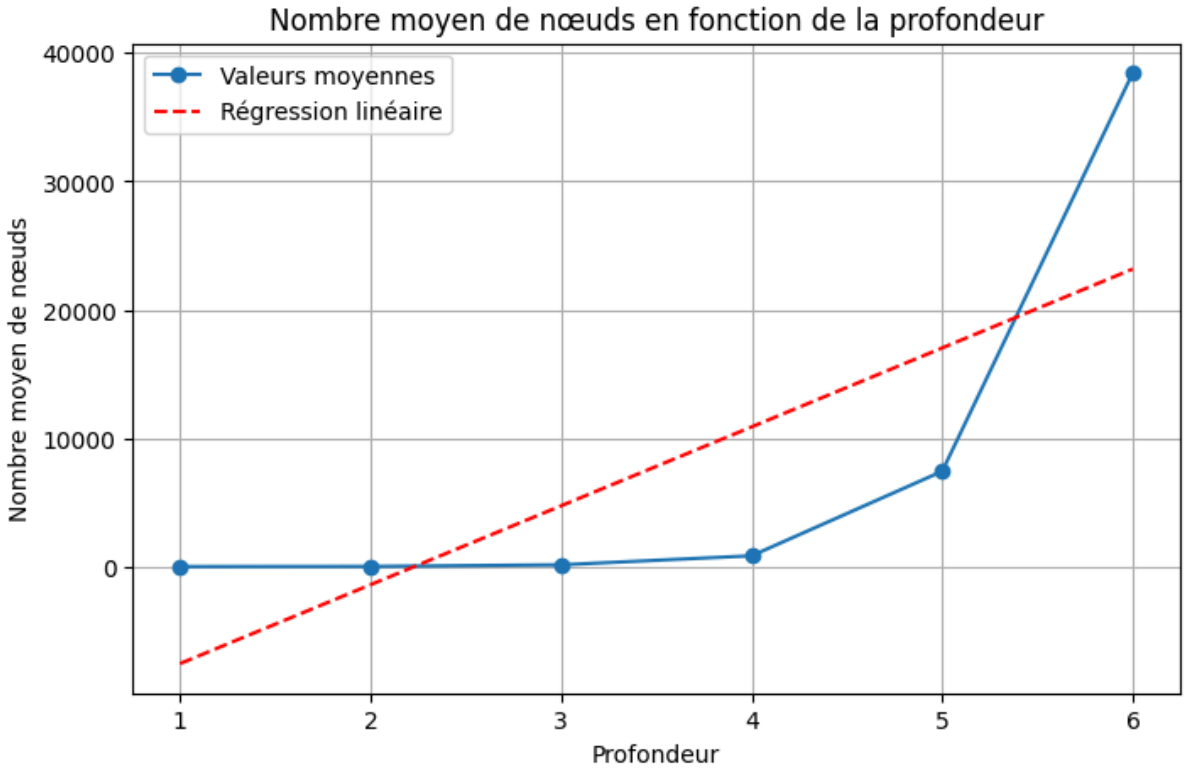


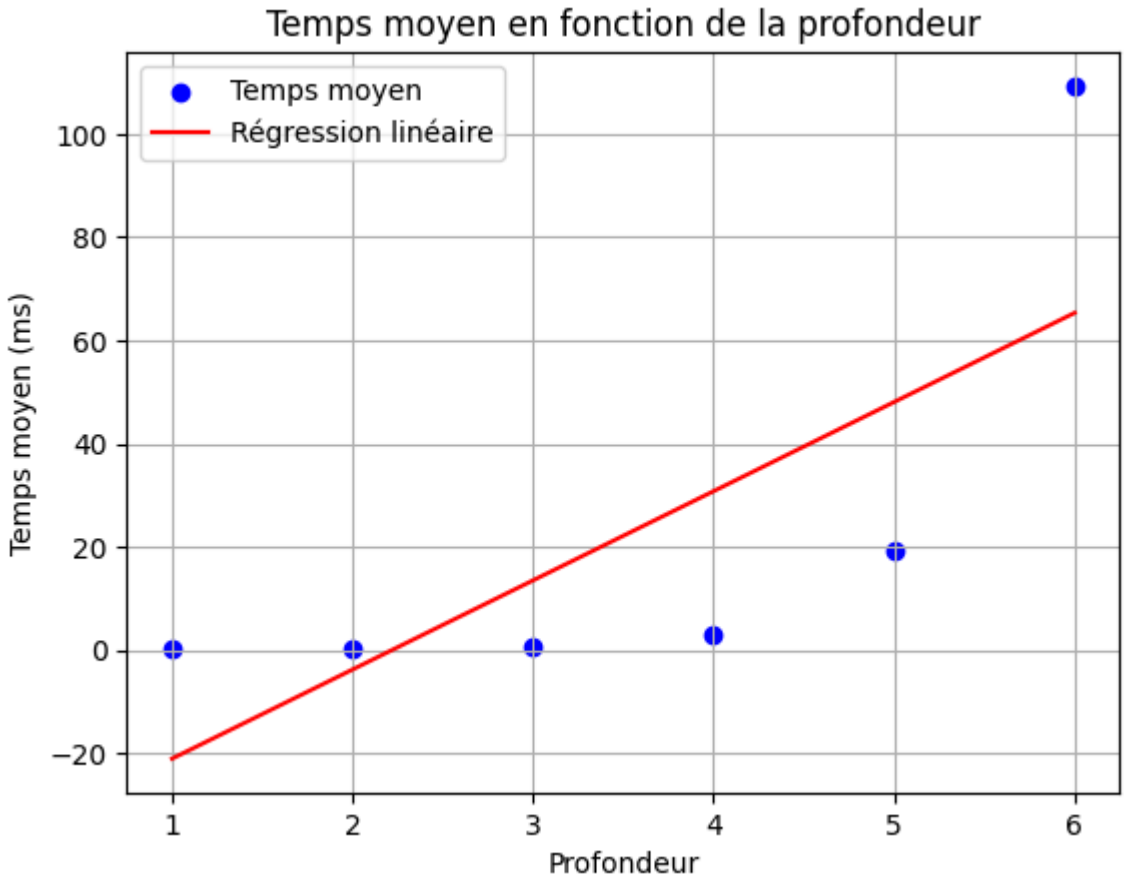


Minimax vs Minimax

Profondeur	Nombre de noeuds moyen	Temps moyen (ms)	Victoires	Vainqueur dominant
1	7	0.370	28%	souvent perdant
2	16	0.146	85%	souvent gagnant
3	140	0.681	42%	souvent perdant
4	843	2.895	57%	souvent gagnant
5	7427	19.446	57%	souvent gagnant
6	38441	109.328	28%	souvent perdant

Tableau 2 : Résultats de l'analyse de l'algorithme Minimax contre lui-même selon différentes profondeurs


Nombre moyen de nœuds en fonction de la profondeur
Valeurs moyennes
Régression linéaire
Nombre moyen de nœuds
40000
30000
20000
10000
0
1
2
3
4
5
6
Profondeur


Temps moyen en fonction de la profondeur
Temps moyen
Régression linéaire
Temps moyen (ms)
100
80
60
40
20
0
−20
1
2
3
4
5
6
Profondeur

2. Comparaison de AlphaBeta avec Minimax

On a confronté AlphaBeta et Minimax en faisant jouer 1000 parties pour chaque profondeur de 1 à 6. Les deux algorithmes utilisent la même profondeur à chaque test. On fait jouer 1000 parties AlphaBeta vs Minimax à profondeur égale puis on calcule le nombre de victoires pour chaque algo et on répète pour chaque profondeur.

Profondeur 1	**Victoires Alphabeta**	100%
	Victoires Minimax	0%
	Matchs nuls	0%
Profondeur 2	**Victoires Alphabeta**	100%
	Victoires Minimax	0%
	Matchs nuls	0%
Profondeur 3	**Victoires Alphabeta**	0%
	Victoires Minimax	100%
	Matchs nuls	0%
Profondeur 4	**Victoires Alphabeta**	100%
	Victoires Minimax	0%
	Matchs nuls	0%
Profondeur 5	**Victoires Alphabeta**	100%
	Victoires Minimax	0%
	Matchs nuls	0%
Profondeur 6	**Victoires Alphabeta**	100%
	Victoires Minimax	0%
	Matchs nuls	0%

Tableau 3 : Résultats des affrontements entre Alpha-Beta et Minimax sur 1000 parties de Puissance 4 avec des profondeurs de 1 à 6.

Interprétation des résultats

On observe clairement que Minimax utilise beaucoup moins de nœuds qu'AlphaBeta ce qui s'explique par leur nature algorithmique différente. Minimax explore l'arbre de jeu de manière exhaustive mais sans optimisation particulière ce qui limite le nombre total de nœuds générés. Cette caractéristique le rend plus efficace sur des profondeurs faibles où sa

simplicité lui permet de terminer les calculs rapidement et de prendre des décisions compétitives.

En revanche, AlphaBeta montre un comportement opposé. Son mécanisme d'élagage, bien que réduisant théoriquement le nombre de nœuds à évaluer, génère en pratique un volume considérable de calculs intermédiaires avec des ordres de grandeur variant entre 10^6 et 10^9 nœuds. Cette complexité le rend plus lent que Minimax dans les configurations basiques. Cependant, son exploration devient un avantage pour des profondeurs plus élevées. La capacité d'AlphaBeta à éliminer des branches non prometteuses (*pruning*) lui permet d'approfondir son analyse stratégique là où Minimax serait rapidement saturé, expliquant sa supériorité dans les parties complexes nécessitant une vision à long terme.

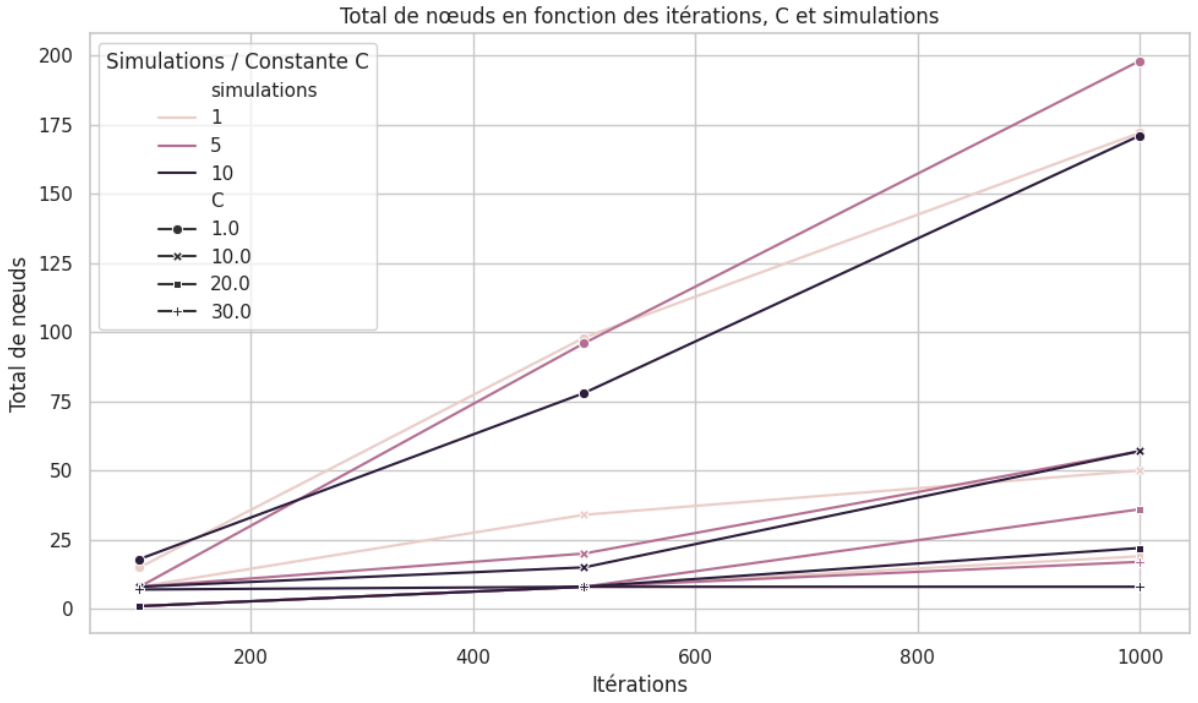
Minimax offre une solution rapide et légère pour des décisions immédiates tandis qu'AlphaBeta, malgré son coût computationnel, devient indispensable lorsque la profondeur d'analyse est déterminante pour la qualité du jeu.

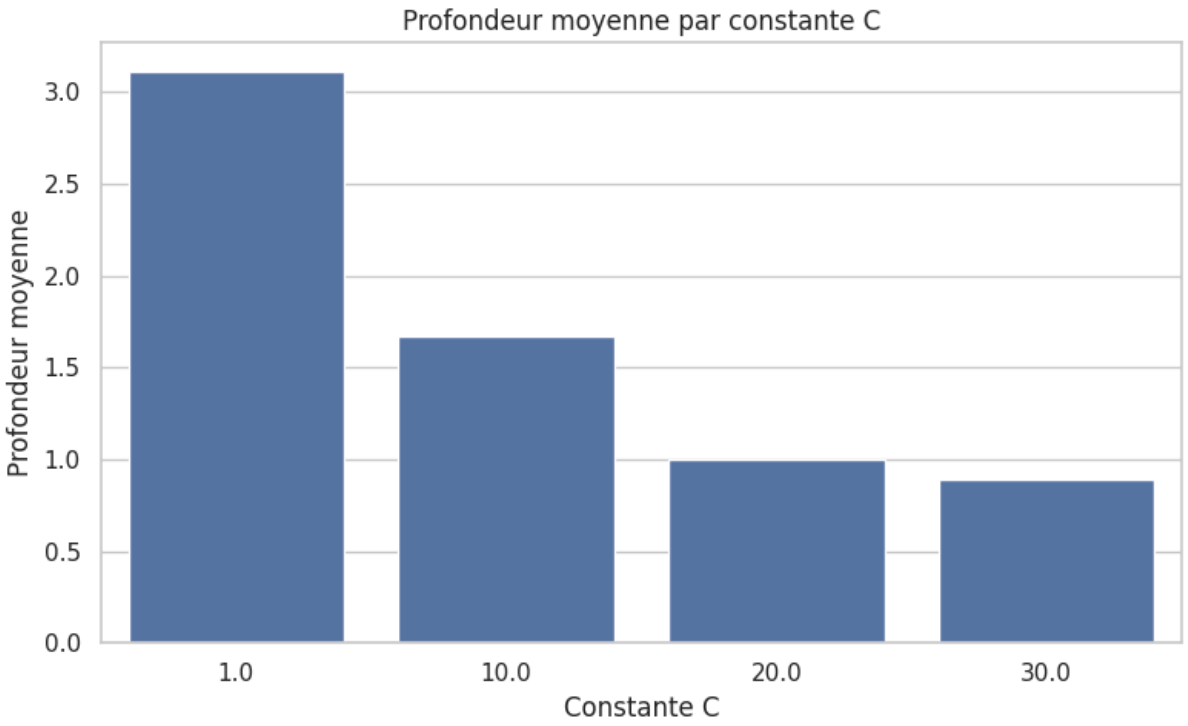
Cependant, dans les deux cas, le nombre de nœuds et le temps d'exécution augmente de manière exponentielle en fonction de la profondeur.

3. Évaluation MCTS

Nous avons évalué l'algorithme MCTS en faisant varier trois paramètres :

- Nombre d'itérations : 100, 500 et 1000
- Constante d'exploration C : 1, 10, 20 et 30
- Simulations par rollout : 1, 5 et 10





Interprétation des résultats

On constate que lorsque C est grand, l'algorithme explore beaucoup de coups différents mais ne va pas en profondeur (largeur > profondeur). Par exemple, il regarde les 7 colonnes possibles au premier niveau et 49 combinaisons au deuxième niveau. À l'inverse, avec un C petit, MCTS se concentre sur les meilleurs coups et creuse plus profondément mais en explorant moins d'options à chaque niveau (ordre de grandeur de 10-100 nœuds contre seulement 1-10 pour C grand). On voit aussi qu'augmenter le nombre de simulations permet d'aller plus loin dans l'arbre même avec le même C.

On déduit que plus C est élevé, plus l'arbre s'étend en largeur (favorise l'exploration), tandis qu'avec un C faible, il se développe en profondeur en se focalisant sur les branches intéressantes (favorise l'exploitation).

4. Comparaison de Alphabeta et Minimax avec MCTS

Après avoir trouvé les meilleurs paramètres pour chaque algorithme pendant nos premiers tests, nous avons fait jouer AlphaBeta et Minimax contre MCTS sur 1000 parties. Nous avons testé trois versions de MCTS : une avec C = 0.5 (<1), une avec C = 2 (plutôt basée sur l'exploitation) et une autre plus "exploratrice" avec C = 20. Cela nous permet de comparer les algorithmes dans de bonnes conditions et de voir comment le paramètre C influence les résultats.

Nous avons choisi :

- Minimax : profondeur 3 (meilleur rapport perf/ressources)
- AlphaBeta : profondeur 6 (assez profond sans être excessif)
- MCTS : 1000 itérations + 5 simulations (optimal) pour comparer équitablement les algos sur 1000 parties.

C = 0.5			C = 2			C = 20		
Victoires Minimax	**Victoires MCTS**	**Matchs nuls**	**Victoires Minimax**	**Victoires MCTS**	**Matchs nuls**	**Victoires Minimax**	**Victoires MCTS**	**Matchs nuls**
10%	90%	0%	54%	46%	0.6%	97%	3%	0%

Tableau 4 : Résultats de l'affrontement entre Minimax et MCTS sur 1000 parties de Puissance 4

C = 0.5			C = 2			C = 20		
Victoires AlphaBeta	**Victoires MCTS**	**Matchs nuls**	**Victoires AlphaBeta**	**Victoires MCTS**	**Matchs nuls**	**Victoires AlphaBeta**	**Victoires MCTS**	**Matchs nuls**
24%	76%	0%	84%	16%	0.6%	99%	1%	0%

Tableau 5 : Résultats de l'affrontement entre AlphaBeta et MCTS sur 1000 parties de Puissance 4

Interprétation des résultats

On constate qu'avec C=0.5, MCTS devient très fort: il bat Minimax dans 90% des cas et AlphaBeta dans 76% des cas. Cela montre qu'en se concentrant davantage sur les bons coups (exploitation), MCTS surpasse les autres méthodes.

Avec C=2, les résultats sont plus équilibrés : MCTS reste compétitif contre Minimax (46% de victoires) mais perd nettement face à AlphaBeta (seulement 16% de victoires). Cela confirme qu'AlphaBeta est plus puissant que Minimax grâce à son élagage intelligent.

Enfin, avec C=20, MCTS échoue complètement (seulement 3% contre Minimax, 1% contre AlphaBeta). Une exploration trop aléatoire le rend inefficace dans le Puissance 4, où les coups tactiques comptent plus que la chance.

MCTS excelle avec un petit C (0.5) car il exploite mieux les stratégies gagnantes, AlphaBeta reste meilleur que Minimax, mais MCTS peut le battre si bien réglé, des constantes C trop grandes sont inutilisables car elles entraînent une exploration trop importante qui nuit aux performances. Le bon réglage de C est donc crucial pour que MCTS soit compétitif.

CONCLUSION

Dans ce projet, nous avons implémenté et testé trois algorithmes d'IA pour le jeu de Puissance 4 : Minimax, Alpha-Beta et MCTS. Nous avons comparé leurs performances à travers plusieurs tests (profondeur, nombre d'itérations, etc.). Minimax et Alpha-Beta donnent de bons résultats mais sont vite limités par la profondeur à cause du temps et de la mémoire. MCTS, plus flexible, permet d'explorer plus rapidement sans tout parcourir mais il dépend beaucoup du nombre d'itérations. Globalement, chaque algorithme a ses avantages selon le contexte mais MCTS reste plus adapté pour des recherches plus profondes sans trop de ressources.

Bibliographie

[1] Puissance4_IA repository: Repository GitHub contenant le code source, les tableaux de résultats et l'ensemble des fichiers utilisés dans notre projet (toutes les instructions indiquant comment l'exécuter sont dans le README)