Rapport de Projet IA

Jeu choisi : 2048

Présenté par

Gwendoline Delestre

Lingyun Zhuang

M1 MIASHS IC

Introduction 3

Modélisation 3

Processus du jeu 4

Heuristiques 5

Algorithme de MinMax 6

Expériences 8

Conclusion 10

Références 11

Introduction

2048 se joue sur une grille de 4x4 cases, avec des tuiles de couleurs et de valeurs variées (mais toujours des puissances de deux). Le but du jeu est de faire glisser des tuiles sur une grille, pour combiner les tuiles de mêmes valeurs et créer ainsi une tuile portant le nombre 2048.

Modélisation

Grille initiale : Une grille de 4x4 cases avec 2 tuiles positionnées aléatoirement avec des valeurs de 2 ou 4.

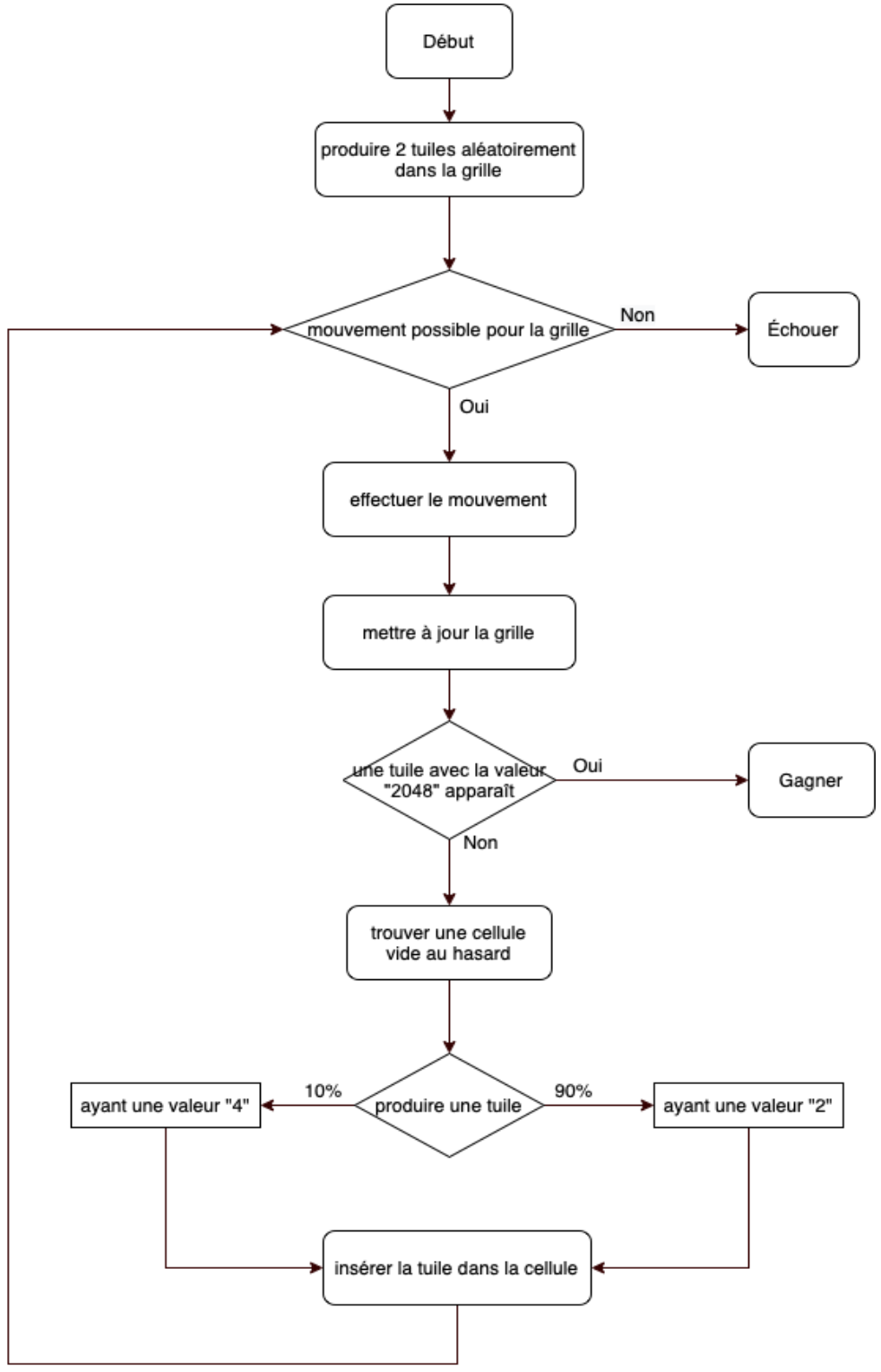
Joueur : Le joueur peut choisir l'une des quatre directions de mouvement : haut, bas, gauche ou droite. Dans certains cas, il y en aura moins de quatre, car il peut y avoir des directions qui ne sont pas disponibles si une colonne ou une ligne comporte déjà 4 tuiles qui ne peuvent pas fusionner. Toutes les tuiles se déplacent dans la direction choisie, et si deux tuiles adjacentes dans cette direction de déplacement ont la même valeur, elles sont combinées en une seule tuile et la valeur de la tuile est doublée.

Ordinateur : Placer une tuile avec une valeur de 2(90%) ou 4(10%) dans une cellule aléatoire qui est actuellement vide.

- Condition gagnante : une tuile avec la valeur 2048 apparaît.
- Condition d'échec : la grille est pleine et il est impossible de se déplacer dans l'une des quatre directions (aucune fusion ne peut être déclenchée).

Processus du jeu

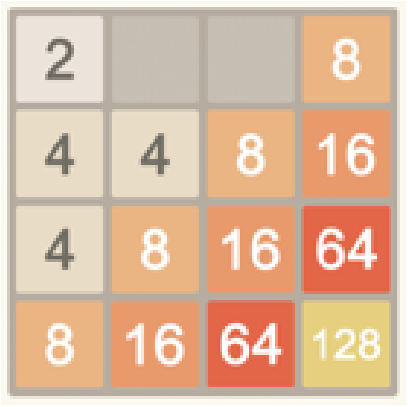
Le jeu commence par générer deux positions aléatoires dans une grille vide avec des valeurs de 2 ou 4. Ensuite, le joueur effectue un mouvement si cela est possible. La grille est mise à jour et si une tuile a une valeur de 2048, le joueur gagne. Sinon, l'ordinateur recherche aléatoirement une cellule vide et insère une tuile de valeur 2 (90% de probabilité) ou de valeur 4 (10% de probabilité). Ensuite, c'est au tour du joueur de se déplacer, et ainsi de suite. Lorsqu'aucun mouvement n'est possible sur la grille, le joueur perd.



Heuristiques

Nous avons trouvé 3 heuristiques pour gagner le jeu.

1. Les tuiles suivent un motif décroissant de droite à gauche et de bas en haut.



2. La différence entre la valeur de chaque tuile et celle de ses voisins est plus faible que possible.

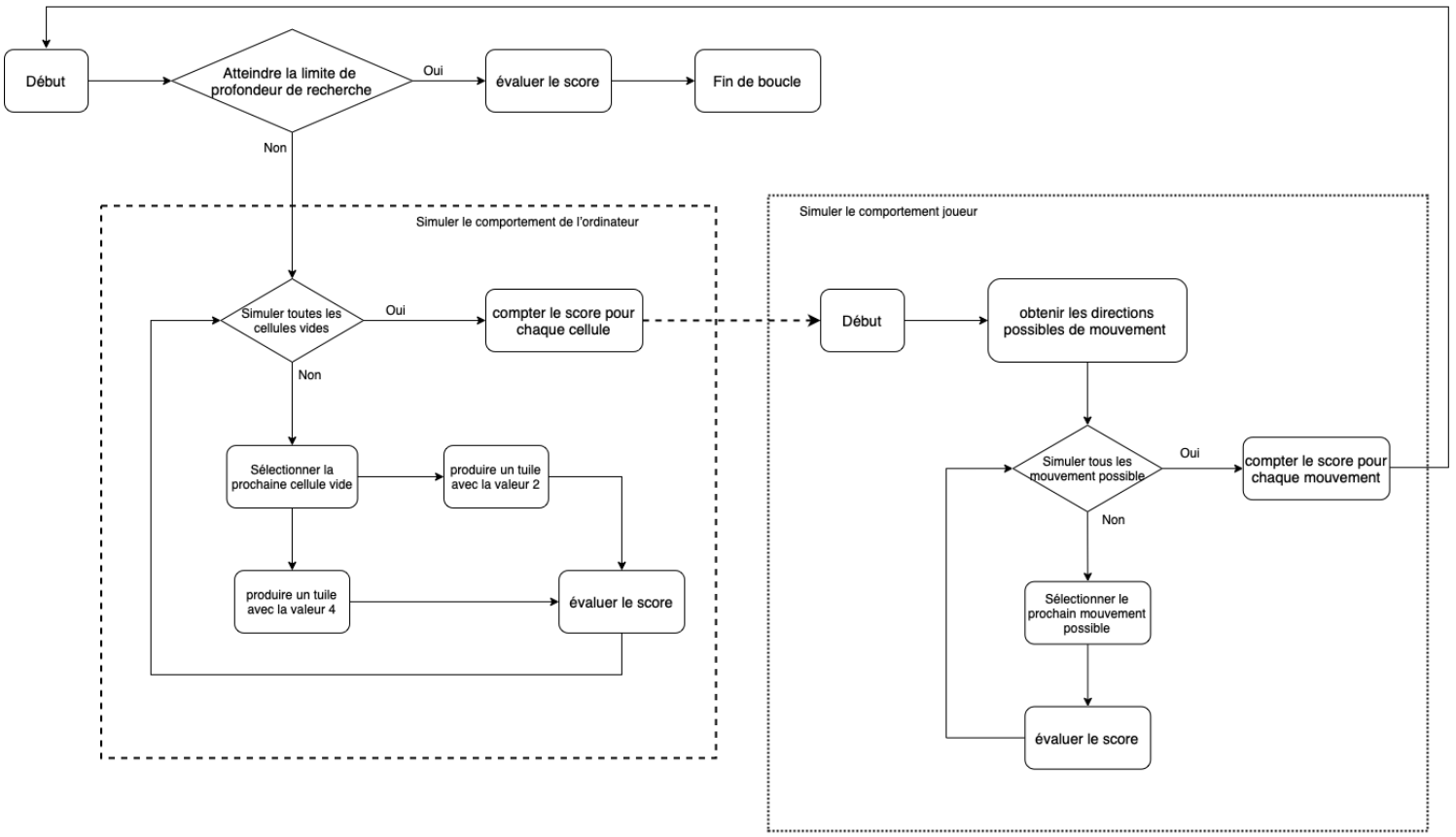
est mieux que

3. Nombre de cellules vides

   Chaque mouvement du joueur doit essayer d'avoir autant de cellules vides sur la grille que possible.

Algorithme de MinMax

La première étape consiste à vérifier si la limite de profondeur de recherche a été atteinte. Si ce n'est pas le cas, le comportement de l'ordinateur est d'abord simulé : des tuiles avec les valeurs 2 et 4 sont insérées sur toutes les cellules vides et le score est ensuite évalué. On simule ensuite le comportement du joueur : simuler le mouvement dans toutes les directions possibles, puis évaluer la grille mise à jour. La boucle est alors exécutée et arrêtée si la limite de profondeur de recherche est atteinte.



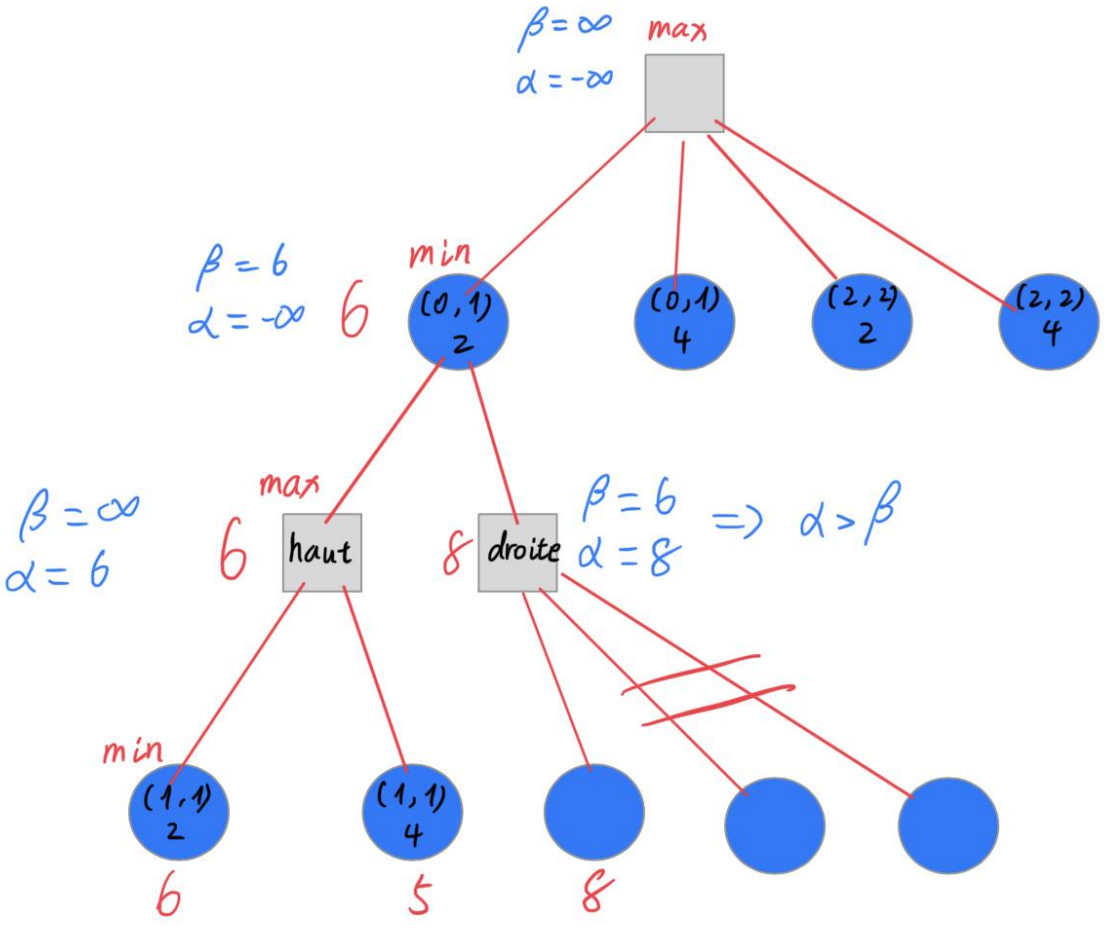
Représentation d'élagage Alpha-beta:

Le jeu 2048 peut essentiellement être abstrait comme un jeu à deux joueurs. Nous avons donc choisi d'utiliser l'algorithme Minimax avec élagage Alpha-beta. L'arbre de jeu représente les coups possibles jusqu'à une certaine profondeur à partir de la position courante. Chaque nœud de l'arbre est évalué avec une fonction d'évaluation.

Le nœud max représente le joueur qui effectue un mouvement sur la grille, il choisit un coup de score maximal parmi ses fils. Le nœud min représente l'ordinateur qui insère une tuile aléatoire, il choisit un coup de score minimal parmi ses fils.

L'élagage alpha-bêta est utilisé afin de réduire le nombre de nœuds évalués par l'algorithme minimax. Le résultat est le même qu'avec Minimax, mais les branches de l'arbre qui ne peuvent pas influencer le résultat final sont ignorées, ce qui permet de gagner du temps de recherche.

Dans l'image, le nœud "haut" prend une valeur maximale parmi ses nœuds fils, ici il est de 6. Son nœud parent est un nœud min, qui prend une valeur minimale parmi ses enfants. Le score maximum (β) de ce nœud est donc de 6. Le score maximum (β) du nœud 'droite' est également de 6, mais la valeur de son premier nœud fils est de 8, ce qui signifie que le score minimum (α) de nœud 'droite' est de 8. Dans ce cas, la valeur minimale du nœud "droite" est supérieure à sa valeur maximale(α > β). La position courante ne peut pas être le résultat du meilleur mouvement pour aucun des deux joueurs, et n'a donc pas besoin d'être explorée plus en avant. Donc ici on effectue une coupure beta.

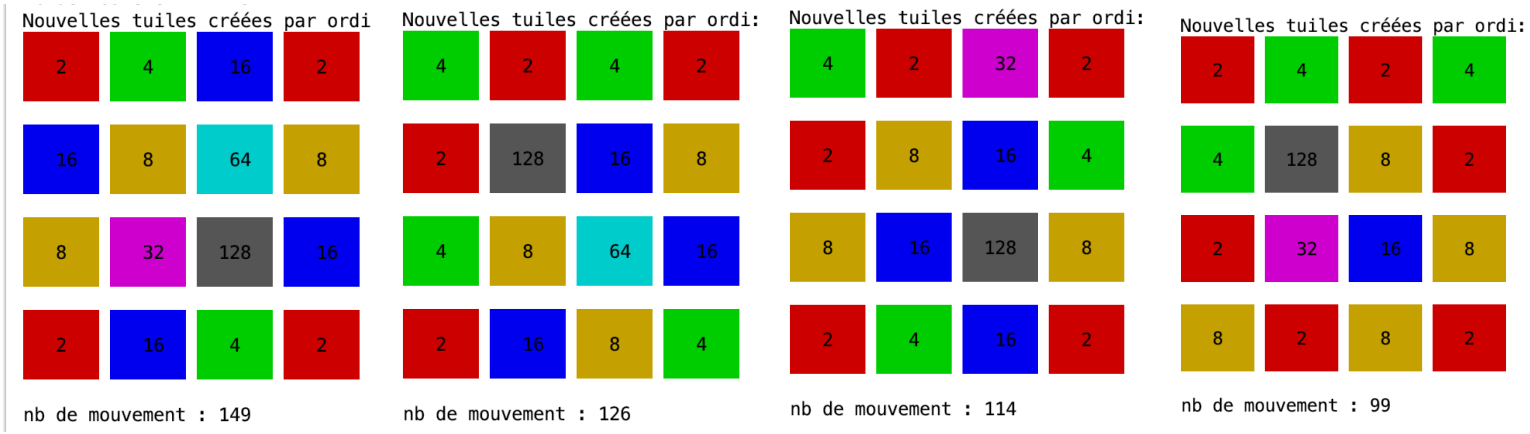


Expériences

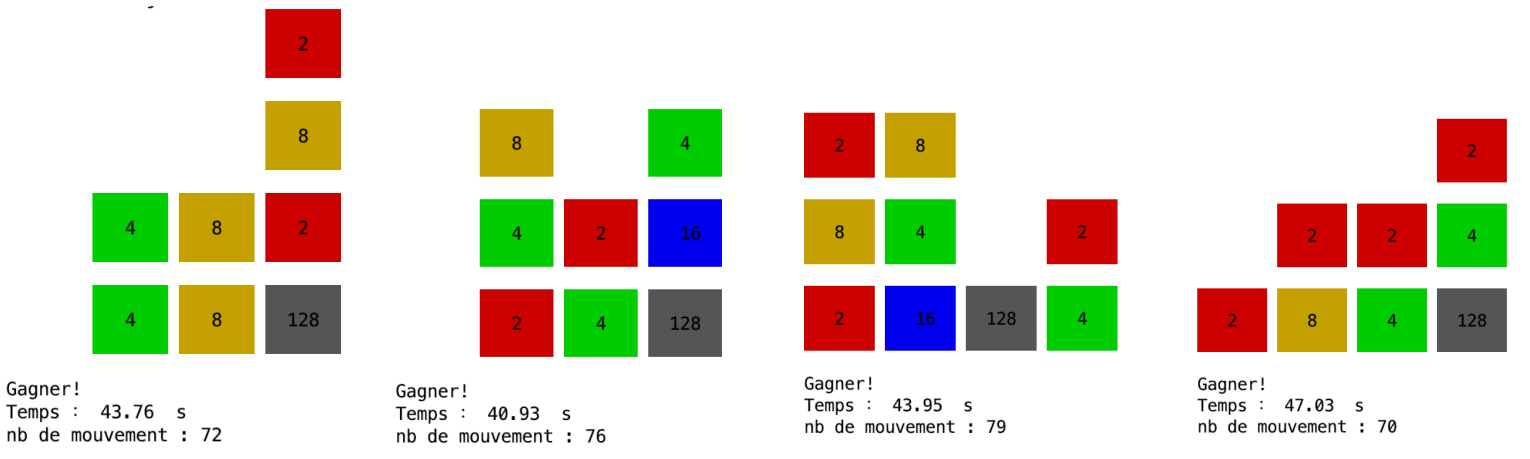
Pour chacune des expériences ci-dessous nous avons pris en compte le temps d'exécution nécessaire lorsque nous affichons la grille de résultats dans la console uniquement, le temps d'exécution de l'interface graphique comprend des paramètres différents et est donc plus long.

Nous avons d'abord testé si un joueur peut obtenir une valeur de 2048 lorsqu'il ne suit qu'un ordre fixe de mouvement vers le haut, le bas, la gauche et la droite.

Pour cela, nous avons fait quatre expériences. Chaque fois que le jeu se terminait (la grille était complètement remplie), la valeur maximale des tuiles n'atteignait que 128. Nous avons également compté le nombre de mouvements et le nombre moyen de mouvements est 122.



Ensuite nous avons fait quatre expériences avec l'algorithme MinMax. Il ne fallait en moyenne que 74 mouvements pour atteindre la valeur de 128. Et nous pouvons voir que la grille est dans un état favorable pour gagner, avec beaucoup de positions libres pour que le jeu continue.



L'étude sur le profondeur de recherche:

Afin d'explorer l'effet de différentes profondeurs de recherche sur les résultats, nous avons fixé la limite de la profondeur de recherche à 4 et 6, et réalisé six expériences distinctes. Nous avons enregistré si notre IA a gagné, le nombre de mouvements et le temps d'exécution dans le tableau ci-dessous.

	profondeur de recherche = 4			profondeur de recherche = 6		
	Gagner ?	nb moves	temps	Gagner ?	nb moves	temps
1	Non(1024)	946	20.95s	Oui	992	205.14s
2	Non(1024)	862	17.82s	Oui	972	257.46s
3	Non(1024)	981	20.24s	Oui	961	312.8s
4	Oui	1010	21.72s	Oui	997	266.73s
5	Non(256)	428	9.12s	Non(1024)	945	252.32
6	Oui	1093	21.22s	Oui	1018	287.11s

Nous pouvons voir que lorsque la limite de profondeur de recherche était de 4, notre IA n'a gagné que deux fois sur six expériences en exécutant un nombre moyen de mouvements de 1051, avec un temps d'exécution moyen de 21,47 secondes. Trois fois, la valeur maximale des tuiles était de 1024, et une fois seulement de 256.

Lorsque la limite de profondeur de recherche était de 6, notre IA a gagné la plupart des expériences (5/6), avec une moyenne de 988 mouvements et un temps d'exécution moyen de 264 secondes.

Ces expériences nous permettent de conclure que plus la recherche est profonde, plus les chances que l'IA réussisse à gagner sont élevées, mais aussi plus le temps d'exécution est long. Lorsque la recherche n'est pas assez profonde, le temps d'exécution pour gagner est court, mais il y a un risque d'obtenir un très mauvais résultat.

L'étude sur l'élagage alpha-bêta :

Afin d'explorer l'effet de l'élagage alpha-bêta sur les résultats, nous avons expérimenté deux versions d'algorithme MinMax avec et sans élagage alpha-beta. Pour gagner du temps dans nos expériences, nous avons fixé la valeur pour gagner le jeu à 128. Nous avons enregistré le nombre de mouvements et le temps d'exécution dans le tableau ci-dessous.

	Sans l'élagage α-β		Avec l'élagage α-β	
	nb moves	temps	nb moves	temps
1	69	119 s	71	40.04 s
2	66	138.49 s	74	39.35 s
3	78	126.83 s	67	41.01 s
4	67	129.57 s	75	33.6 s

Nous pouvons observer que les temps de déplacement moyens des deux versions sont très proches (70 et 71.75), ce qui indique que l'élagage alpha-beta n'affecte pas les résultats de l'algorithme minmax. Cependant, le temps d'exécution moyen de la version avec élagage alpha-beta (38.5s) est bien inférieur à celui de l'autre version (128.47s). Cela indique que l'élagage alpha-beta permet d'économiser beaucoup de temps de recherche et de calcul.

Conclusion

D'après les expériences effectuées, nous avons vu qu'il est plus rapide d'avoir une profondeur de recherche faible pour obtenir rapidement la fin du jeu cependant il est très peu probable de gagner. Une profondeur de recherche plus importante permet d'obtenir de meilleurs résultats en dépit du temps d'exécution. Enfin, utiliser l'algorithme de MinMax avec élagage permet de diminuer le temps d'exécution.

Références

Description du jeu : https://fr.wikipedia.org/wiki/2048_(jeu_vidéo)

Dans notre code de programme, nous avons référencé la structure et certaines des fonctions d'un projet sur GitHub :
https://github.com/SrinidhiRaghavan/AI-2048-Puzzle