ANNÉE 2022 - 2023

3250.3 Intelligence artificielle I
TP5 – Rapport technique – ISC3il-b

Algorithme MiniMax – Othello

Développer une intelligence artificielle pour le jeu Othello, à l'aide de l'algorithme MiniMax.

Étudiants participant à ce travail :

Nicolas Aubert, ISC3il-b
Théo Vuilliomenet

Présenté à :

Fabrizio Albertetti

Restitution du rapport : **06.01.2023**

Période : **2022 – 2023**

École : **HE-Arc, Neuchâtel**

haute école arc ingénierie
neuchâtel berne jura www.he-arc.ch

Table des matières

1 - Introduction

1.1 - Contexte

Ce travail pratique rentre dans le cadre du cours *3250.3 Intelligence artificielle I*, dispensé par M Albertetti.

Le jeu Othello est un jeu de stratégie populaire qui se joue sur un plateau de 8x8 cases. Le but du jeu est de terminer la partie avec le plus grand nombre de pions de sa couleur sur le plateau. Dans ce projet, nous allons développer une intelligence artificielle capable de jouer à Othello de manière autonome en utilisant l'algorithme minimax.

L'algorithme minimax est un algorithme de recherche d'arbres qui permet à une intelligence artificielle de déterminer la meilleure action à effectuer dans un jeu à deux joueurs en évaluant toutes les possibilités de jeu à chaque tour. Nous allons implémenter cet algorithme pour créer une intelligence artificielle qui peut jouer contre un adversaire humain, contre elle-même ou contre d'autres intelligences.

Notre intelligence artificielle sera capable de prendre en compte différents facteurs tels que la mobilité, la position des pions sur le plateau et la possibilité de créer des chaînes de pions pour maximiser ses chances de victoire. Nous espérons que ce projet permettra non seulement de créer une intelligence artificielle redoutable pour le jeu Othello, mais aussi de mieux comprendre comment mettre en œuvre l'algorithme minimax dans un contexte de jeu à deux joueurs.

2 - Réalisation

2.1 - Le jeu Othello

2.1.1 - Règles

Le jeu Othello est un jeu de stratégie pour deux joueurs qui se joue sur un plateau de 8x8 cases. Chaque joueur a un jeu de pions d'une couleur, généralement noire et blanche.

Au début de la partie, quatre pions sont placés au centre du plateau, deux blancs et deux noirs, disposés afin qu'ils forment une croix. Les joueurs jouent chacun leur tour, en plaçant un pion de leur couleur sur le plateau, dans le but d'entourer au moins un pion adverse.

Lorsqu'un pion entoure un ou plusieurs pions adverses, ceux-ci sont retournés et prennent la couleur du joueur qui a joué le coup. Par exemple, si un joueur joue un pion blanc de manière à ce qu'il entoure un pion noir, le pion noir est retourné et devient blanc.

2.1.2 -Conditions de victoires

Le jeu se termine lorsqu'un joueur ne peut plus jouer de coup valide, ou lorsque le plateau est plein. Le gagnant est le joueur qui a le plus grand nombre de pions de sa couleur sur le plateau. Si le nombre de pions est égal, la partie est déclarée nulle.

2.2 - L'algorithme MiniMax

L'algorithme minimax est un algorithme de recherche d'arbres utilisé pour trouver la meilleure action à effectuer dans un jeu à deux joueurs. Il est souvent utilisé pour développer des intelligences artificielles pour les jeux de stratégie tels que les échecs, le go, et l'othello.

2.2.1 - Fonctionnement

L'algorithme commence par générer un arbre de recherche qui représente toutes les possibilités de jeu à chaque tour. Chaque nœud de l'arbre représente une position de jeu, et chaque branche représente une action possible. Voici un diagramme en arbre illustrant ce principe :

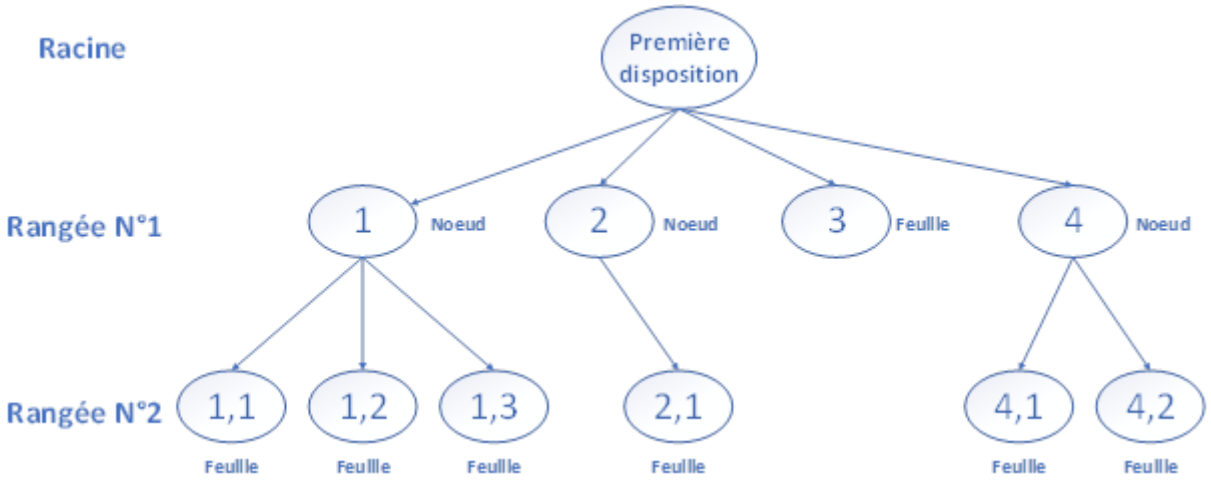

Figure 1 : Exemple d'un schéma en arbre sur lequel se base l'algorithme Minimax

Le premier nœud représente la disposition initiale, qui est la racine de l'arbre (le début). Elle se trouve en haut du diagramme. Sur la rangée numéro un se trouvent toutes les possibilités engendrées par le premier nœud (première disposition). Il en va de même pour les rangées suivantes où se trouvent les dispositions engendrées par les nœuds de la rangée précédente.

Chaque nœud qui ne donne pas suite à d'autres nœuds, appelé feuille, par exemple si la partie est terminée, est affecté d'un score. Une fois toutes les possibilités calculées et toutes les feuilles affectées par un score, l'algorithme parcourt l'arbre de bas en haut afin de faire remonter un score à la racine. Ce score représente alors le meilleur choix à faire par rapport à la situation initiale.

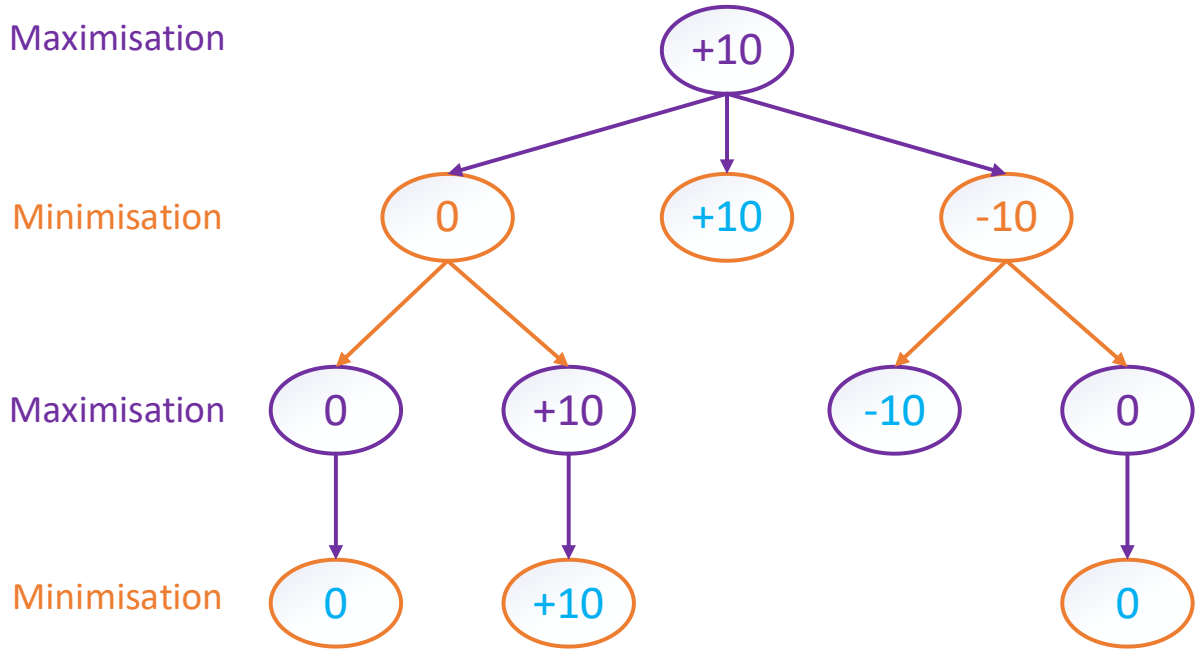

Figure 2 : Exemple d'un schéma en arbre simplifié avec les attributions de scores par l'algorithme Minimax

Sur le schéma ci-dessus, l'algorithme remonte l'arbre en propageant les valeurs de chaque feuille vers les nœuds parent. Pour chaque nœud, l'algorithme choisit la valeur minimale (pour le joueur adverse) ou la valeur maximale (pour le joueur courant) parmi les valeurs de ses enfants, selon le tour de jeu représenté par le nœud.

2.3 - L'élagage alpha-bêta

Afin d'optimiser l'algorithme, il est possible de réduire le nombre de cas à calculer : c'est le rôle de l'élagage alpha-bêta.

Le principe est simple : intervenir dans l'algorithme de Minimax sans en changer le résultat. Pour ce faire, il faut évaluer les nœuds de l'arbre où les valeurs ne sont pas utiles.

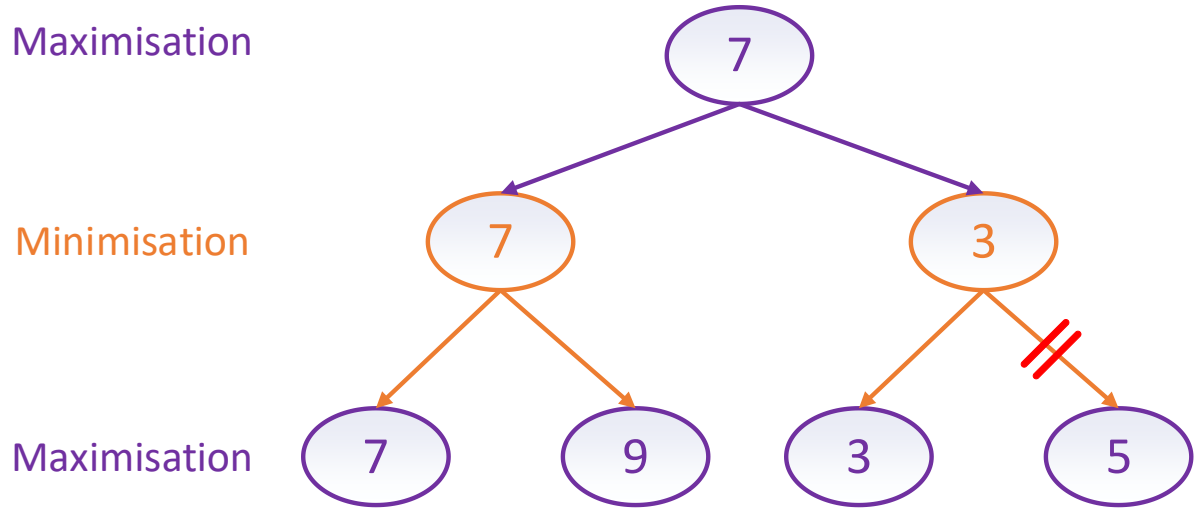

Figure 3 : Exemple de schéma en arbre avec l'élagage alpha-bêta

Sur ce schéma en arbre, la branche de gauche possède les valeurs 7 et 9 sur sa dernière rangée. Comme la ligne d'au-dessus minimise, elle choisira donc le 7.

Sur la branche de droite, la rangée centrale, qui minimise, choisira le 3 avant de calculer la valeur du nœud de droite (ici 5 sur le schéma). Mais comme elle minimise, elle ne changera de valeur seulement si celle-ci est inférieure à 3. C'est pourquoi il est alors inutile de calculer cette valeur : la première ligne maximise les résultats, elle prendra de toute façon la valeur 7.

7 est plus grand que 3, et le nœud avec la valeur 3 ne peut que diminuer son score.

2.4 - Fonction d'évaluation

2.4.1 -Mobilité

La mobilité d'un joueur se réfère au nombre de coups valides qu'il peut jouer pendant son tour. Plus un joueur a de mobilité, plus il a de possibilités de jouer et donc de contrôler le plateau. La mobilité est donc un facteur important à prendre en compte lors de l'évaluation d'une position sur le plateau.

La mobilité potentielle, en revanche, se réfère à la mobilité qui pourrait être obtenue par un joueur s'il était en mesure de jouer plusieurs coups consécutifs. Par exemple, si un joueur peut jouer un coup qui lui donnera la possibilité de jouer un deuxième coup immédiatement, sa mobilité potentielle est deux. La mobilité potentielle peut être importante à prendre en compte dans une stratégie de jeu à long terme, car elle peut donner une idée de la capacité d'un joueur à prendre le contrôle du plateau à mesure que la partie avance. La mobilité potentielle n'a pas été prise en compte dans notre fonction d'évaluation, par manque de temps et de motivation.

Pour calculer le score lié à la mobilité du joueur courant, la formule suivante a été utilisée :

$$100 \cdot \frac{nb\ coups\ possibles\ pour\ le\ joueur\ courant - nb\ coups\ possibles\ pour\ l'autre\ joueur}{nb\ coups\ possibles\ pour\ le\ joueur\ courant + nb\ coups\ possibles\ pour\ l'autre\ joueur}$$

Elle permet d'obtenir un score de mobilité pour le joueur courant, en fonction de la mobilité de l'autre joueur. Cette valeur varie entre -100 et 100.

Par exemple, si le joueur courant peut jouer sept coups différents, alors que l'autre joueur ne peut en jouer que trois, le score vaudra :

$$100 \cdot \frac{7-3}{7+3} = 100 \cdot \frac{4}{10} = 40$$

Inversement, si le joueur courant ne peut jouer que six coups différents, et alors que l'autre peut en jouer treize, son score vaudra :

$$100 \cdot \frac{6-13}{6+13} = 100 \cdot \frac{-7}{19} \cong -36.8$$

2.4.2 - Contrôle du plateau

En cherchant sur internet différentes idées de stratégies pour Othello, nous sommes rapidement tombés sur l'importance du placement et la distinction entre les différentes régions du plateau de jeu. Nous avons notamment trouvé le schéma ci-dessous pour une grille de 8x8.

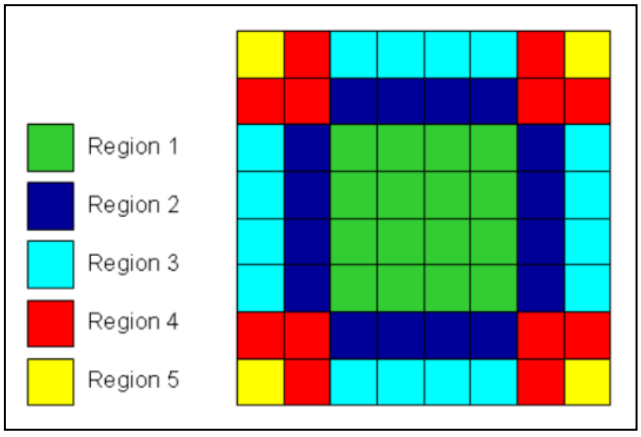

Figure 4: Schéma des régions 8x8

C'est à ce moment que nous avons eu l'idée de pondérer les zones de la grille afin de l'utiliser comme fonction d'évaluation. Pour que les pondérations soient facilement modifiables, nous avons imaginé un système de score qui va jusqu'à 100. Il suffit ensuite de répartir ces 100 points entre les différentes zones pour obtenir notre grille pondérée.

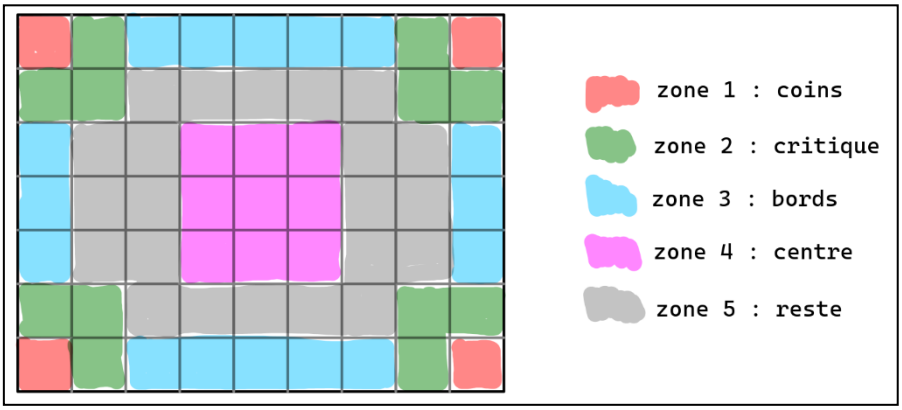

Figure 5: schéma des régions 7x9

Voici les pondérations qui ont été utilisées en fonction des différentes zones :

Zone 1	Zone 2	Zone 3	Zone 4	Zone 5
50%	0%	25%	15%	10%

Figure 6 : Tableau illustrant les pourcentages en fonction des zones du plateau

Nous avons commencé par donner des pondérations arbitraires et ensuite nous avons testé différentes pondérations. Nous n'avons pas uniquement assigné des pondérations aléatoires aux différentes zones, mais nous avons aussi essayé de créer une certaine stratégie par le biais de ces pondérations.

Par exemple nous avons toujours donné beaucoup d'importance aux coins, car ils sont primordiaux pour une victoire. Nous avons aussi toujours donné 0% à la zone critique qui entoure les coins, car il faut absolument éviter de prendre ces cases au risque de voir l'adversaire nous voler le coin par la suite. Pour ce qui est des trois autres zones, nous avons simplement essayé de nombreuses possibilités et sommes arrivés à une certaine répartition qui donnait de bons résultats.

2.4.2.1 Génération de grilles pondérées

Afin de pouvoir rapidement modifier les pondérations des zones et de tester la nouvelle grille pondérée, nous avons rapidement codé une petite fonction qui s'occupe de calculer tout ça pour nous. Le principe est très simple nous définissons une première grille de 7x9 à zéro.

```
grid = [
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0]
]
```

Ensuite nous définissons une deuxième grille pour les zones.

```
zones = [
    [1, 2, 3, 3, 3, 3, 3, 2, 1],
    [2, 2, 5, 5, 5, 5, 5, 2, 2],
    [3, 5, 5, 4, 4, 4, 5, 5, 3],
    [3, 5, 5, 4, 4, 4, 5, 5, 3],
    [3, 5, 5, 4, 4, 4, 5, 5, 3],
    [2, 2, 5, 5, 5, 5, 5, 2, 2],
    [1, 2, 3, 3, 3, 3, 3, 2, 1]
]
```

Il suffit encore de renseigner les pondérations, le nombre de cases par zone et le score total afin tout calculer automatiquement. Finalement, le résultat est une grille dans laquelle toutes les cellules sont pondérées en fonction de leur zone.

```
score_array = [
    [12.5, 0.0, 1.406, 1.406, 1.406, 1.406, 1.406, 0.0, 12.5],
    [0.0, 0.0, 0.454, 0.454, 0.454, 0.454, 0.454, 0.0, 0.0],
    [1.406, 0.454, 0.454, 1.944, 1.944, 1.944, 0.454, 0.454, 1.406],
    [1.406, 0.454, 0.454, 1.944, 1.944, 1.944, 0.454, 0.454, 1.406],
    [1.406, 0.454, 0.454, 1.944, 1.944, 1.944, 0.454, 0.454, 1.406],
    [0.0, 0.0, 0.454, 0.454, 0.454, 0.454, 0.454, 0.0, 0.0],
    [12.5, 0.0, 1.406, 1.406, 1.406, 1.406, 1.406, 0.0, 12.5]
]
```

2.4.3 - Possession des coins

La possession de coins est très importante car elle permet de contrôler une grande partie de la grille de jeu. Les coins sont difficiles à capturer par l'adversaire, donc une fois qu'un joueur a pris possession d'un coin, il peut être difficile pour l'autre joueur de le récupérer. De plus, avoir un coin permet de contrôler les lignes et les colonnes adjacentes, ce qui peut être très avantageux pour le joueur qui détient le coin. C'est pourquoi la possession de coins est généralement considérée comme très importante dans l'évaluation de l'état d'une partie d'Othello.

Pour prendre en compte cet aspect dans notre fonction d'évaluation, nous comptons le nombre de coins possédés par le joueur actuel, ainsi que pour le l'autre joueur. La pondération de cette possession est alors calculée avec la formule suivante :

$$100 \cdot \frac{nb\ coins\ possédés\ par\ le\ joueur\ courant - nb\ coins\ possédés\ par\ l'autre\ joueur}{nb\ coins\ possédés\ par\ le\ joueur\ courant + nb\ coins\ possédés\ par\ l'autre\ joueur}$$

Elle permet d'obtenir un score de possession des coins pour le joueur courant, en fonction du nombre de coins possédés par l'autre joueur. Cette valeur varie entre -100 et 100.

Par exemple, si le joueur courant possède deux coins, et que l'autre joueur n'en possède qu'un, la pondération vaudra alors :

$$100 \cdot \frac{2-1}{2+1} = 100 \cdot \frac{1}{3} = 33,\overline{3}$$

Voici un tableau résumant les différentes valeurs que peut prendre cette fonction :

Nb coins joueur courant	Nb coins autre joueur	Score
0	0	undefined (0)
0	1	-100
0	2	-100
0	3	-100
0	4	-100
1	0	100
1	1	0
1	2	-33,33333333
1	3	-50
2	0	100
2	1	33,33333333
2	2	0
3	0	100
3	1	50
4	0	100

Figure 7 : Tableau résumant les valeurs possibles pour le score de possession des coins

2.4.4 -Possession des pièces

La possession de pièces est importante dans l'évaluation de l'état d'une partie d'Othello car elle peut avoir un impact sur la stratégie de jeu et sur les options disponibles pour chaque joueur. Généralement, plus un joueur a de pièces, plus il a de possibilités de jouer et plus il peut exercer de pression sur l'adversaire.

En fin de partie, la possession de pièces peut être particulièrement importante, car il y a généralement moins de mouvements possibles et il peut être plus difficile de capturer des pièces adverses. Dans ces situations, le nombre de pièces possédées par chaque joueur peut être un bon indicateur de qui a l'avantage dans la partie.

Pour calculer le score lié à la possession du nombre de pièces, la formule suivante a été utilisée :

$$100 \cdot \frac{nb\ pièces\ possédées\ par\ le\ joueur\ courant - nb\ pièces\ possédées\ par\ l'autre\ joueur}{nb\ pièces\ possédées\ par\ le\ joueur\ courant + nb\ pièces\ possédées\ par\ l'autre\ joueur}$$

Elle permet d'obtenir un score de possession des pièces pour le joueur courant, en fonction du nombre de pièces possédées par l'autre joueur. Comme toutes les fonctions de calculs de score, celle-ci possède un domaine d'arrivée dans l'intervalle -100 et 100.

La valeur 100 sera retournée si le joueur courant possède toutes les pièces du plateau, et -100 si c'est le cas pour le second joueur.

3 - Résultats

Ces tests ont été effectués sur un échantillon de 100 simulations, avec une profondeur limitée à trois, afin de réduire leur durée.

3.1 - Adversaire aléatoire

Voici un tableau résumant les résultats obtenus avec notre intelligence artificielle, contre un adversaire jouant des coups aléatoires. Il est précisé dans la partie de gauche quel(s) élément(s) (mobilité, possession des coins, possession des pièces, contrôle du plateau) ont été utilisés pour les simulations.

Éléments utilisés dans la fonction d'évaluation				**Pourcentage de victoire sur 100 parties**
Mobilité	**Nombre de coins**	**Nombre de pièces**	**Contrôle du plateau**	
			X	93
		X		84
		X	X	94
	X			90
	X		X	92
	X	X		97
	X	X	X	**99**
X				77
X			X	83
X		X		80
X		X	X	86
X	X			83
X	X		X	92
X	X	X		90
X	X	X	X	95

Figure 8 : Tableau des résultats, contre un adversaire jouant aléatoirement

Nous avons constaté que la mobilité, pourtant prometteuse de premier abord, semble altérer les résultats. C'est pourquoi, dans notre implémentation, nous avons décidé de retirer cet élément afin d'obtenir les meilleurs résultats possibles.

La fonction d'évaluation de notre implémentation finale utilise alors :

- La possession des coins,
- La possession des pièces,
- Ainsi que le contrôle du plateau.

En effet, comme illustré dans le tableau ci-dessous, c'est en prenant en compte uniquement ces trois éléments que les résultats sont les plus satisfaisants.

4 - Améliorations / optimisations potentielles

4.1 - Mobilité potentielle

Bien qu'expliquée dans la section « Réalisation », le principe n'a pas été pris en compte dans notre implémentation (uniquement la mobilité simple). Celle-ci permettrait peut-être d'obtenir de meilleurs résultats qu'avec uniquement la mobilité, qui altérait beaucoup les résultats.

4.2 - Réflexion sur la fonction d'évaluation

Après avoir implémenté une fonction d'évaluation complexe faisant intervenir de nombreux aspects comme la parité, la stabilité, la mobilité et la capture des coins, mais aussi l'avancement de la partie, nous avons eu envie de tester une fonction d'évaluation bien plus simple. N'utiliser que la grille pondérée comme fonction d'évaluation.

Nous sommes partis du constat que les aspects les plus importants étaient presque tous représentés par la grille pondérée. C'est pourquoi nous avons tenté de ne jouer qu'avec la grille pondérée comme fonction d'évaluation et de laisser alpha bêta minimiser et maximiser pour nous afin de créer sa propre stratégie pour gagner.

Jusque-là, nous avons essayé d'inculquer une stratégie à notre IA par le biais de la fonction d'évaluation complexe décrite plus haut. Mais ne serait-il pas plus intéressant de laisser la possibilité à alpha bêta de créer sa propre stratégie grâce à la grille pondérée ?

Toutes nos sources parlent de l'importance des aspects décrits plus haut, mais ici le contexte est différent, car premièrement nous ne jouons pas avec la même grille, mais une grille 7x9 et nous devons concevoir une IA qui utilise alpha bêta.

Finalement, les résultats étaient mitigés, mais ça restait très intéressant de voir à quel point la grille pondérée toute seule comme fonction d'évaluation face à notre autre fonction bien plus complexe et lourde en calcul pouvait rivaliser. De plus, la grille pondérée est très efficace face à l'adversaire aléatoire.

5 - Annexes

5.1 - Table des illustrations

5.2 - Bibliographies et références

Principes stratégiques | Fédération Française d'Othello. (s. d.). https://www.ffothello.org/othello/principes-strategiques/

A. (2022, 17 octobre). *Othello Strategy – Top 10 Tips to Win at Othello/Reversi*. Board Game Country. Consulté le 4 décembre 2022, à l'adresse https://www.boardgamecountry.com/othello-strategy/

wikiHow. (2020, 14 juin). *Simple Ways to Win Othello : 10 Steps (with Pictures)*. Consulté le 6 décembre 2022, à l'adresse https://www.wikihow.com/Win-Othello

Sannidhanam, V. & Annamalai, M. (s. d.). *An Analysis of Heuristics in Othello*. University of Washington Computer Science & Engineering. Consulté le 4 décembre 2022, à l'adresse https://courses.cs.washington.edu/courses/cse573/04au/Project/mini1/RUSSIA/Final_Paper.pdf

Othello Evaluation Function. (2012b, septembre 8). Stack Overflow. Consulté le 12 décembre 2022, à l'adresse https://stackoverflow.com/questions/12334216/othello-evaluation-function

Buro, M. (2002, 2 novembre). *An Evaluation Function for Othello Based on Statistics*. SkatGame. Consulté le 17 décembre 2022, à l'adresse https://skatgame.net/mburo/ps/evalfunc.pdf

Geiger, B. (2020, 24 décembre). *PFP Final Project Report : Othello and Minimax*. cs.columbia.edu. Consulté le 20 décembre 2022, à l'adresse http://www.cs.columbia.edu/~sedwards/classes/2020/4995-fall/reports/Othello.pdf

Aubert, N. & Druart, N. (2021, 2 février). *Intelligence artificielle : Implémenter une intelligence artificielle pour le jeu du morpion avec l'algorithme Minimax*. HE-Arc, Neuchâtel.