Matthew DeVille
Rémi Dadvisard

Compte Rendu de Projet 2i013

JEUX A DEUX JOUEURS

Partie I: Conception des jeux

Un fichier **game.py** contient les fonctions communes aux deux programmes, pour alléger les codes et faciliter l'accès aux différentes fonctions de base utiles pour l'Awélé comme pour l'Othello:

getCopieJeu(jeu)	addCoupJoue(jeu, coup)
finJeu(jeu)	getCoupsJoues(jeu)
coupValide(jeu, c)	resetCoupsValides(jeu)
saisieCoup(jeu)	getCoupsValides(jeu)
joueCoup(jeu,coup)	getScores(jeu)
initialiseJeu()	getJoueur(jeu)
getGagnant(jeu)	changeJoueur(jeu)
tirets(s)	getScore(jeu,joueur)
affiche(jeu)	setCaseVal(jeu, ligne, colonne, val)
getPlateau(jeu)	getCaseVal(jeu, ligne, colonne)

Ces fonctions sont décrites en commentaire sur le code dans game.py,

Toutes ces fonctions utilisent une liste **jeu**, initialisée comme ceci:
[**Plateau** (Tableau d'entiers à deux dimensions),
Joueur(entier égal à 1 ou 2),
Liste des Coups Valides (Liste de paires d'entiers),
Liste des coups joués (Liste de paires d'entiers),
Score (tableau de deux entiers)]

A partir de là, les deux jeux sont codés à l'aide des fichiers **othello.py** (Dossier Othello) et **awele.py** (Dossier Awele) avec des fonctions spécifiques à chaque jeu.
- des fonctions d'initialisation du jeu (**initscore**, **initplateau**),

- une série de fonctions qui ont pour but de déterminer les coups valides (**encadre**, **encadrement, entourageVide, coups** pour Othello et **advaff** pour Awele)
des fonctions pour jouer (**Distribue**, **case_prec**, **case_suiv** et **JoueCoup** pour Awele et uniquement **JoueCoup** pour Othello),

- Une fonction **finJeu** permettant de décider si la partie peut être considérée terminée.

Ensuite les deux dossiers (Awele et Othello) contiennent chacun leur **main.py** qui fait appel à awele(respectivement othello).py et à game.py pour initialiser un jeu, et incrémenter les scores à chaque tour à l'aide d'une boucle while faisant appel à la fonction **finJeu**

Partie II: Implémentation des joueurs

Le dossier Awele/Joueurs contient tous les joueurs implémentés pour Awele et le dossier Othello/Joueurs contient ceux implémentés pour Othello. Chaque joueur a au moins une fonction **saiseCoup** qui renvoie le coup que le joueur décide de jouer à ce tour, et auquel le main fait appel.

Le Joueur Humain:

Le joueur humain tient en quelques lignes: un appel à la fonction **affiche** de game.py pour afficher le plateau, un affichage des coups valides et avec **raw_input** on demande à l'utilisateur de choisir là où les coordonnées du coup choisi.

Avec ces éléments, on a deux jeux qui fonctionnent avec deux joueurs humains.

Les Joueurs Autonomes:

Pour tester les premiers joueurs autonomes, on a créé un **joueur_agauche.py (Awélé)** et un **joueur_premier_coup.py (Othello)** qui jouent toujours respectivement le coup valide disponible au début du plateau et le premier coup dans la liste retournée par getCoupValide. Un joueur aléatoire a aussi été codé. la but de ses trois joueurs est principalement de tester les algorithmes de joueurs autonomes utilisés plus tard:
On doit s'assurer que ces joueurs soient systématiquement battus par nos algorithmes.

Algorithmes Min/max et Alpha/Bêta:

Le principe des deux joueurs créés est le suivant: ils parcourent des arbres de possibilités a l'aide de plusieurs fonctions avec une variable **prof** qui limite le nombre de niveaux explorés pour éviter des surcharges de mémoire, ils ont trois fonctions dont le principe est similaire.

evaluation(jeu): le but de cette fonction est de déterminer l'état du jeu a la fin du parcours, elle utilise les scores (et le nombre de points « stratégiques » que le joueur possède en Othello)

estimation(jeu, coup, p): cette fonction prends en compte un coup et parcours l'arbre jusqu'à la profondeur indiquée dans **prof,** la fonction est récursive et p est incrémenté à chaque appel. Son but est de renvoyer un score d'utilité pour un coup.

decision(jeu, coups): cette fonction prends la liste des coups valide et fait appel à estimation pour chacun d'entre eux pour décider du meilleur coup possible.

L'algorithme Min-max (**joueur_minmax.py**) parcours pour chaque coup possible, toutes les parties qui peuvent en découler. lors de chaque noeud exploré, l'algorithme détermine si il est en train d'analyser un coup qu'il va jouer (noeud Max) ou un coup que l'adversaire va jouer (Noeud Min). Si il est en noeud Max, il doit choisir le coup qui lui permettra

d’obtenir maximum de point, si il est en noeud Min il doit choisir le coup qui lui rapportera le minimum de points, se plaçant dans la situation de l’adversaire.

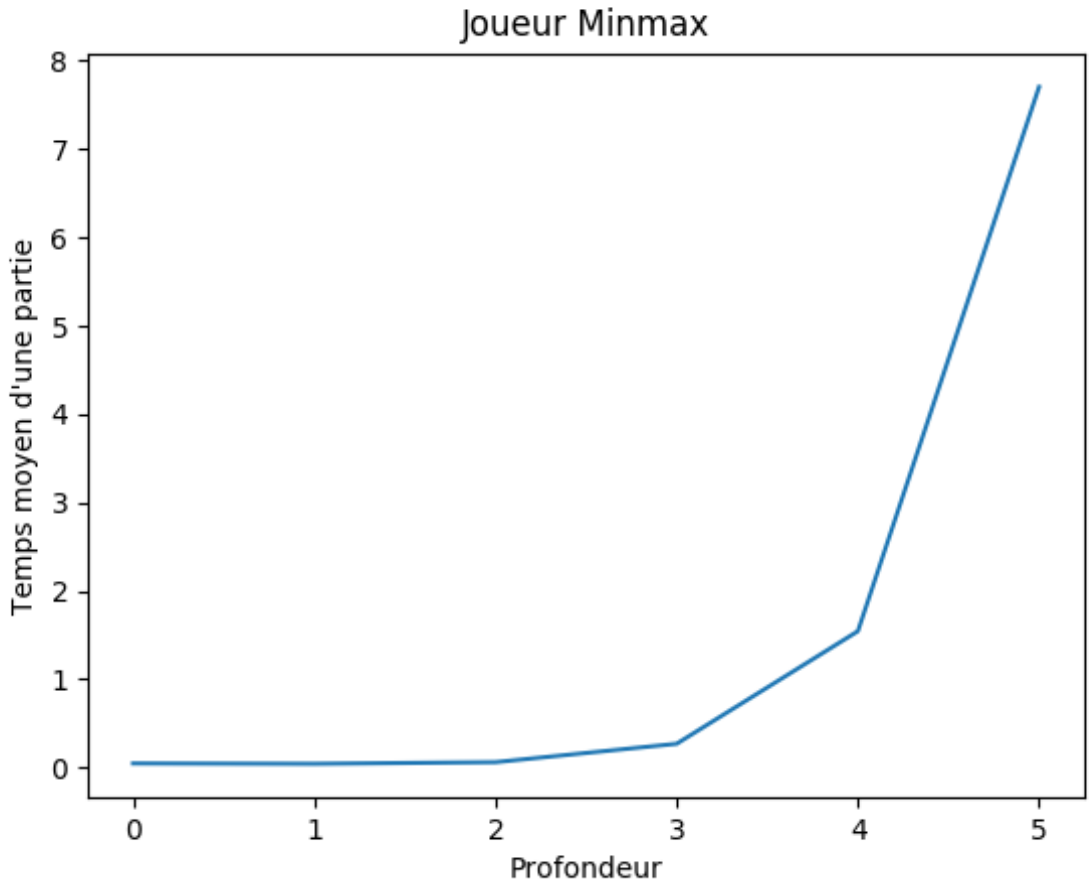
L’algorithme Alpha-bêta (**joueur_alphabeta.py**) est une version de l’algorithme min-max partant du principe que l’algorithme n’a pas à s’intéresser à certaines parties de l’arbre, par exemple pour un noeud max, il n’aura pas a explorer les noeuds qui ne seront pas joués par le noeud min suivant car trop peu avantageux pour ce dernier. Autrement dit si un coup implique que le coup noeud min joué soit improbable, l’algorithme ne le prendra pas en compte. ce qui permet de réduire le nombre de noeuds explorés de l’arbre et donc de calculer le coup avec une meilleure profondeur.

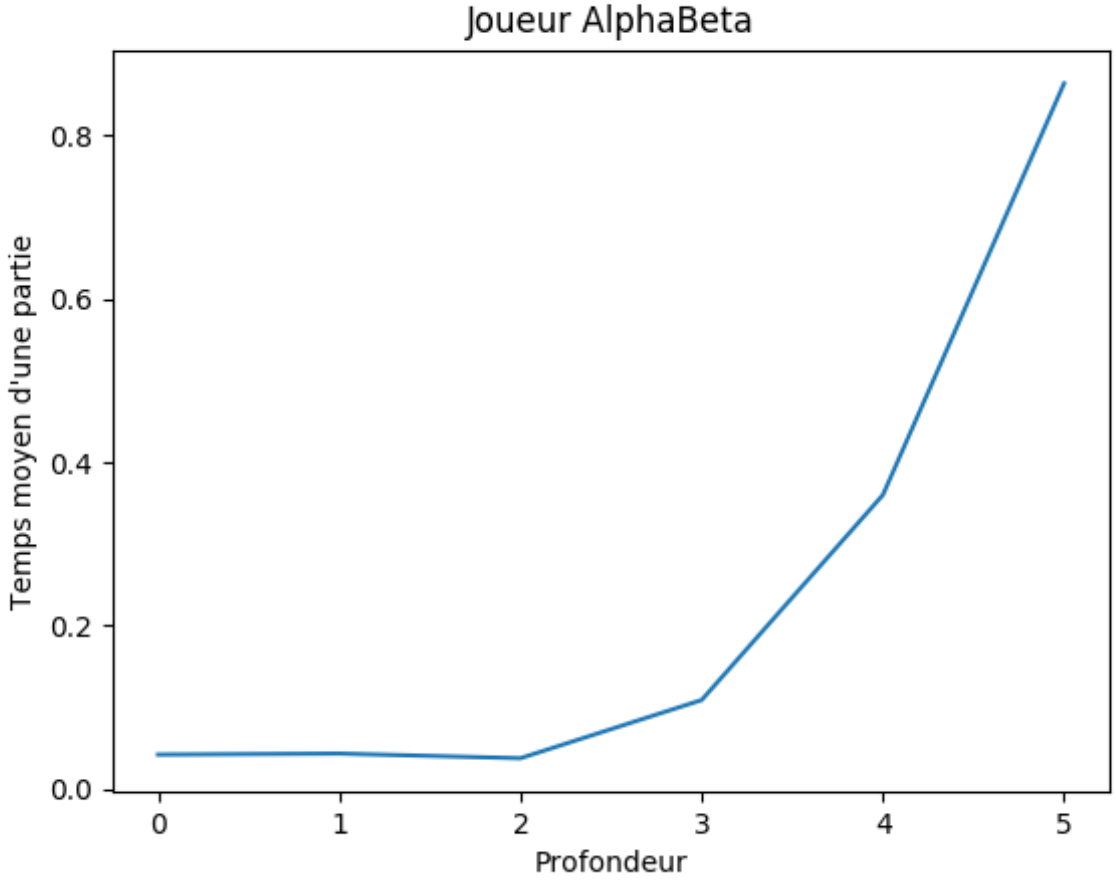
Il est utile pour l’Othello de trier les coups avant de les présenter à l’algorithme, de façon à ce que l’alpha-bêta prennent en premier des coups suffisamment bons et ainsi permettre un élagage plus rapide et efficace. Le site officiel fédération française d’Othello propose de donner la priorité aux coups dans les coins, puis aux coups autour des coins et enfin aux coups sur les côtés.

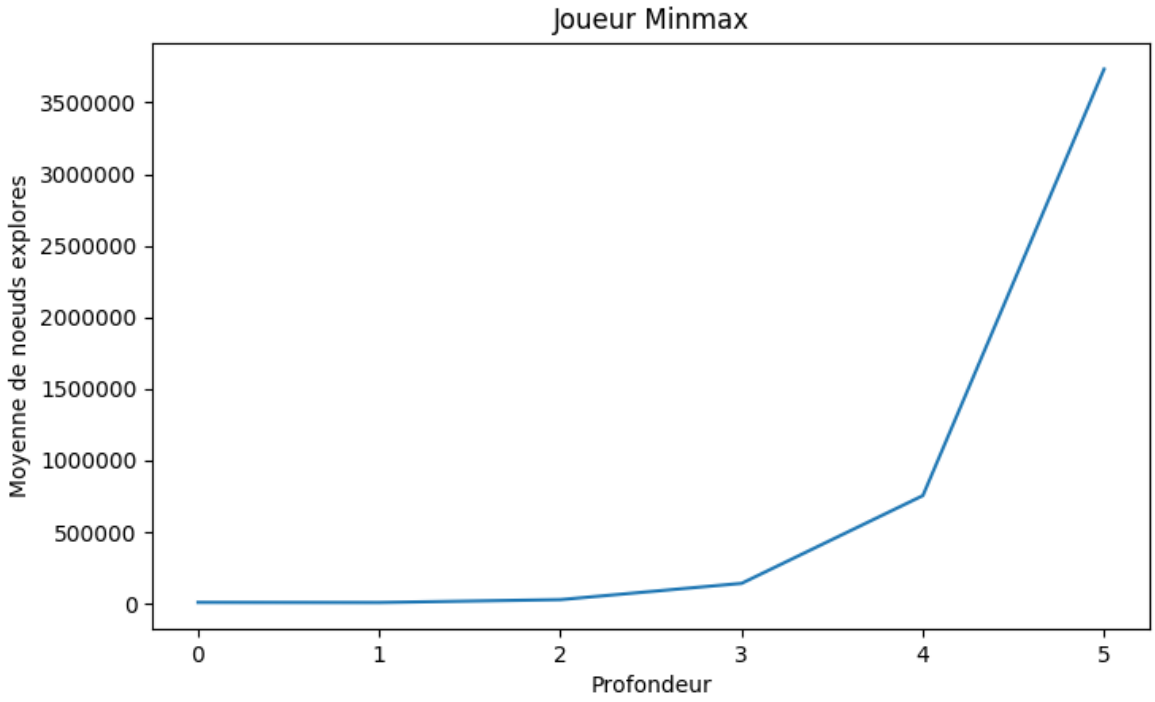
ls fonctions **trieCoups(coups)** et **trieCoupsInverse(coups)** servent a présenter en premier les meilleurs coups pour les noeuds (respectivement max et min).

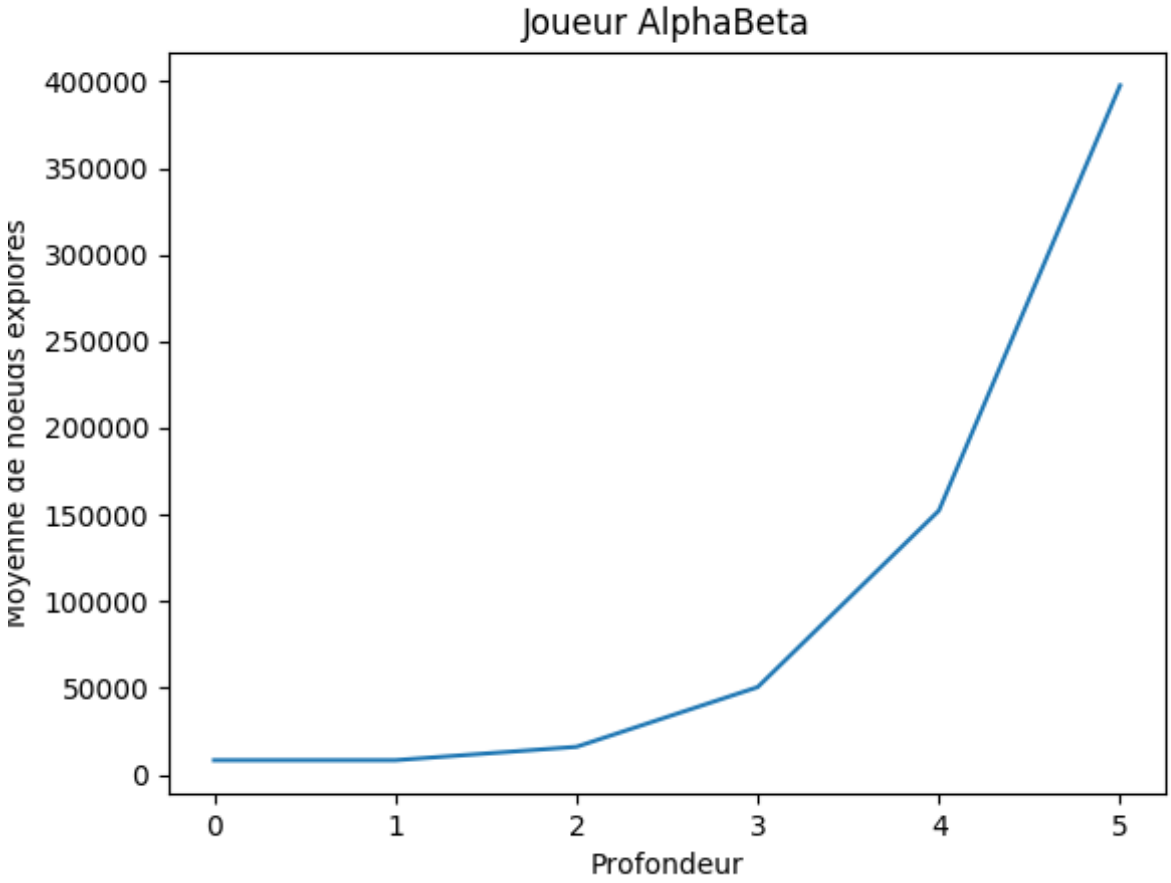
Graphiques (page suivante):

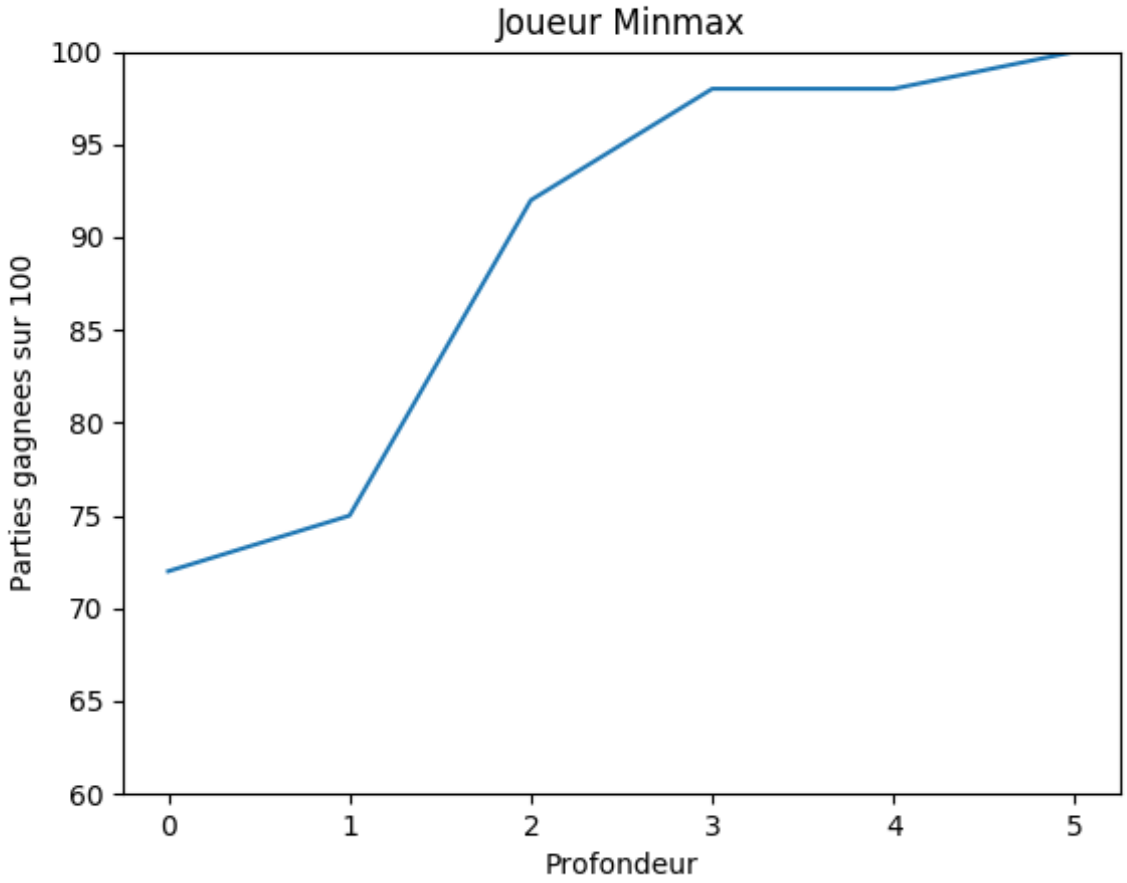
Algorithme MinMax: Pour avoir un joueur étalon a affronter, les parties de tests sont contre un joueur de type Premier Coup Valide

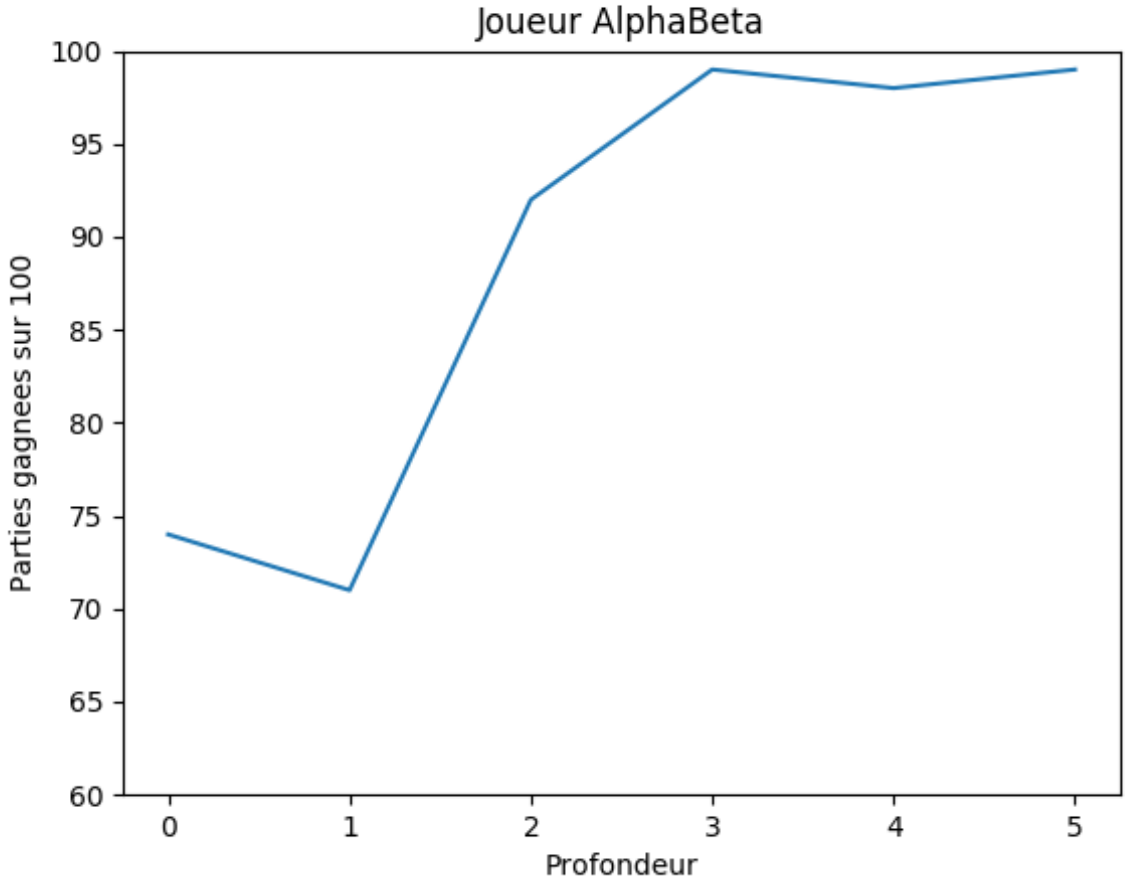

Joueur Minmax
Temps moyen d'une partie
Profondeur
0
1
2
3
4
5
6
7
8


Joueur AlphaBeta
Temps moyen d'une partie
Profondeur
0.0
0.2
0.4
0.6
0.8
0
1
2
3
4
5


Joueur Minmax
Moyenne de noeuds explores
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
0
1
2
3
4
5
Profondeur


Joueur AlphaBeta
Moyenne de noeuds explores
400000
350000
300000
250000
200000
150000
100000
50000
0
0
1
2
3
4
5
Profondeur


Joueur Minmax
Parties gagnees sur 100
Profondeur
100
95
90
85
80
75
70
65
60
0
1
2
3
4
5


Joueur AlphaBeta
Parties gagnees sur 100
Profondeur
100
95
90
85
80
75
70
65
60
0
1
2
3
4
5

Partie III : Joueur par apprentissage

Après avoir testé les joueurs Alphabeta et Minmax, nous nous sommes rendus compte de la lenteur de jeu après une profondeur supérieure à 6. Une des solutions est d'utiliser un joueur par apprentissage supervisé qui aura pour but de simuler un joueur Alphabeta à forte profondeur.

Implémentation :

On utilisera deux fichiers ici. Un pour l'entrainement, nommé trainer.py, l'autre étant le joueur lui même.
Le joueur ressemble majoritairement à un joueur Alphabeta, sauf pour une fonction, ainsi que ses fonction sous jacentes : la fonction evaluation.
Ici la fonction evaluation sera découpée en une somme de sous fonctions qui ont pour but de donner une valeur à certaines caractéristiques du jeu, comme le poids de certaines cases à l'othello.
La somme est sous la forme : a1 * f1(jeu) + a2 * f2(jeu) + … + an * fn(jeu) où a* sont les coefficients à déterminer lors de l'entrainement.

Trainer.py :

L'entrainement consiste à faire jouer en boucle des parties sous la forme suivante :

- Un joueur adversaire, qui ne sert pas vraiment à grand chose à part faire varier les parties
- Un oracle (ou enseignant) qui aura pour but de jouer la partie en même temps que le joueur supervisé
- Le joueur intelligent que l'on souhaite améliorer

De chaque partie il faudra réajuster les différents coefficients a* de l'élève et réessayer