2I013 – Groupe 2 : Programmation d'un jeu à 2 joueurs

Antoine Cadiou (3670631) – Vincent Jouve ()

I. Rendu

- Awele et Othello 100% fonctionnels en local, avec le même comportement que ceux sur le serveur.

Joueurs présents pour chacun des 2 jeux :

- Joueur Humain
- Joueur Premier Coup Valide
- Joueur Horizon 1
- Joueur Minmax
- Joueur Alpha-Beta

Fonctions d'évaluation :

Fonction d'estimation (Algorithme de l'αβ) :

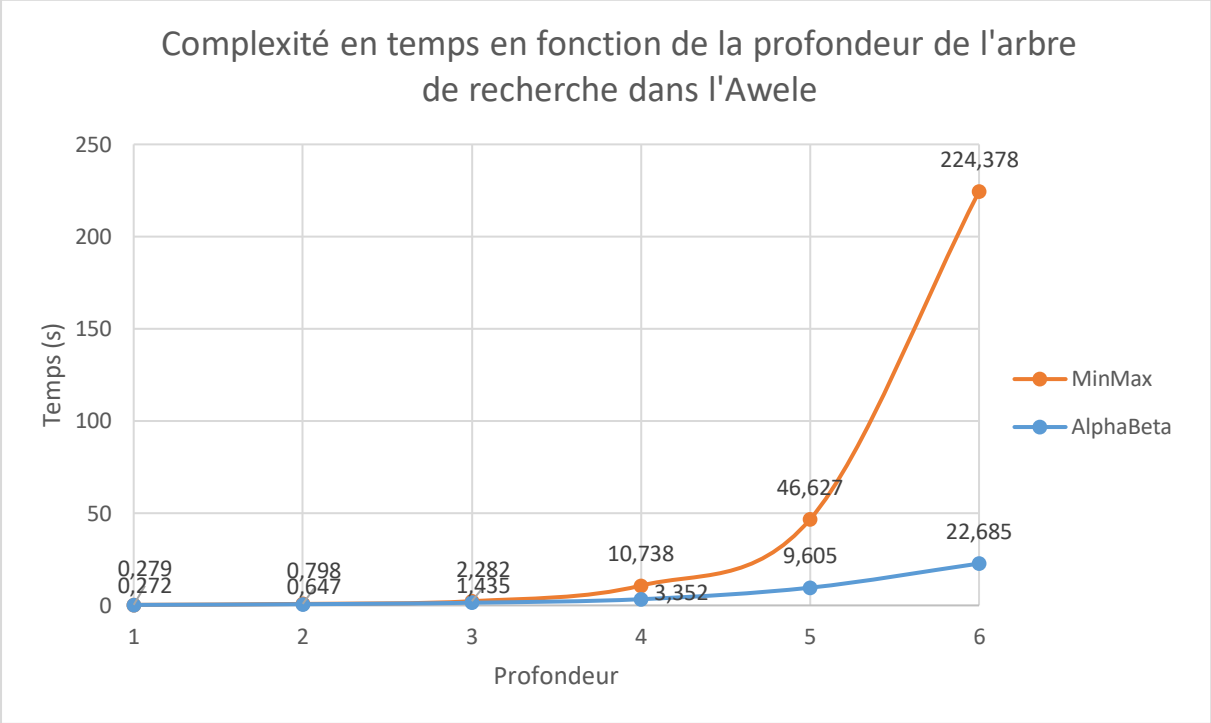
```
def estimation(jeu, profondeur, joueur, alpha, beta):
    #On teste si on arrive à une feuille de l'arbre de recherche
    if (profondeur == 0 or game.finJeu(jeu)):
        return evaluation(jeu, joueur)

    listeCoups = game.getCoupsValides(jeu)
    if(joueur == jeu[1]):
        #max
        maxVal = MIN
        for c in listeCoups:
            saveJeu = game.getCopieJeu(jeu)
            game.joueCoup(saveJeu, c)
            val = estimation(saveJeu, profondeur-1, joueur, alpha, beta)
            maxVal = max (val, maxVal)
            if(beta <= maxVal):
                return maxVal
            alpha = max(alpha,maxVal)
        return maxVal
    else:
        #min
        minVal = MAX
        for c in listeCoups:
            saveJeu = game.getCopieJeu(jeu)
            game.joueCoup(saveJeu, c)
            val = estimation(saveJeu, profondeur-1, joueur, alpha, beta)
            minVal= min (val, minVal)
            if(alpha >= minVal):
                return minVal
            beta = min(beta,minVal)
        return minVal
```

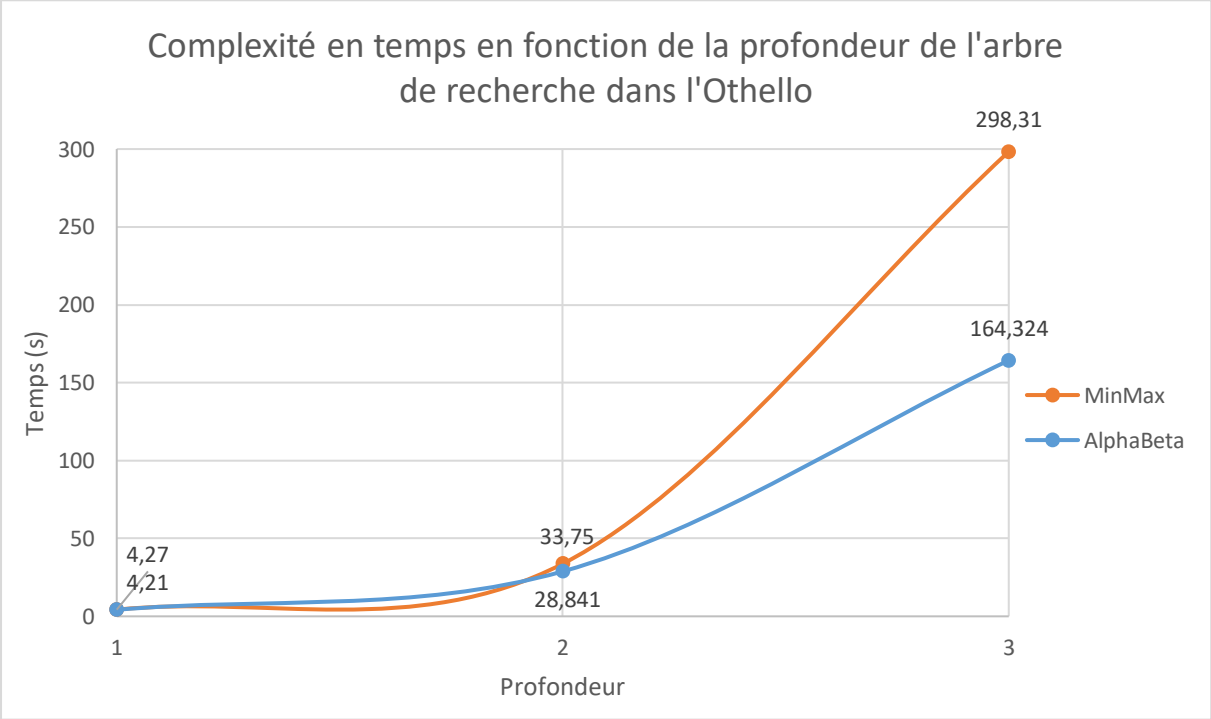
Apprentissage des coefficients :

II. Graphiques

Awele



Othello



III. Tentatives d'améliorations

Awele

- Ajout d'un tri de la liste des coups valides de telle sorte à optimiser le plus possible les élagages de l'αβ.
  C'est-à-dire : dans les coups majorants(minorants), trier du plus favorable(défavorable) au plus défavorable(favorable). Cela afin de trouver l'α(β) le plus rapidement possible.
  Ainsi, nous avons tenté plusieurs tris :

- Tri en fonction de si l'égrainage se termine ou non du côté adverse, par nombre de graines déposées) (ex : un coup qui se termine à la case la plus loin du coté adverse peut potentiellement rapporter plus de points qu'un coup qui se termine au milieu du camp adverse, mais ce dernier reste quand même plus intéressant qu'un coup qui se finit dans notre camps)

Résultats: méthode fonctionnelle mais trop lente pour faire gagner du temps car le tri est effectué à chaque nœud de l'arbre de recherche et a une complexité non négligeable.

- Tri en fonction du nombre de graines à égrainer.

Résultats: fonctions plus rapide d'exécutions mais le nombre de graines n'est pas spécialement prometteur pour la suite de l'élagage, donc aucun gain de temps

Othello

L'algorithme de l'αβ utilisé pour l'Othello est le même que pour l'Awele, cependant le nombre de nœuds à explorer augmente considérablement, et on atteint dès la profondeur 2 la limite (Time Out) du serveur.
Ainsi nous avons jugés très intéressant dans ce jeu de trier les coups pour l'αβ.

- Tri en fonction du 'poids' d'une case par rapport à un masque

```
[[100, -20, 10, 5, 5, 10, -20, 100],
 [-20, -50, -2, -2, -2, -2, -50, -20],
 [10, -2, -1, -1, -1, -1, -2, 10],
 [5, -2, -1, -1, -1, -1, -2, 5],
 [5, -2, -1, -1, -1, -1, -2, 5],
 [10, -2, -1, -1, -1, -1, -2, 10],
 [-20, -50, -2, -2, -2, -2, -50, -20],
 [100, -20, 10, 5, 5, 10, -20, 100]]
```

Résultats: ajout fonctionnel, qui ne fait pas perdre du temps et qui n'en fait pas gagner non plus dû à la faible profondeur (profondeur 3). Cependant sur une profondeur de 5 on passe de 20 minutes à 14 min, ce qui est plus appréciable.

- Recherches et initiation à l'algorithme du Monte-Carlo qui peut permettre d'augmenter le taux de victoires par rapport à un αβ Profondeur 2.
  Avorté: d'après un travail de l' « Edith Cowan University » , le Monte-Carlo ne devient intéressant contre un profondeur 3 (63% de victoires) que si l'on simule 3000 coups pour chaque décision. Dans notre cas, cela mettrait beaucoup trop de temps.