Rapport de projet

[LU2IN013] : Projet de développement

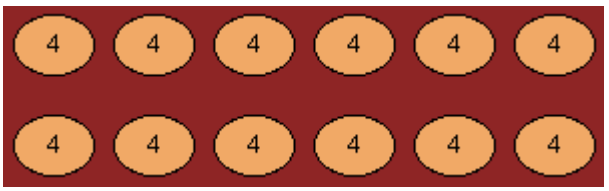
Charles Lin 2020/2021 Corentin Benedetti

Sommaire

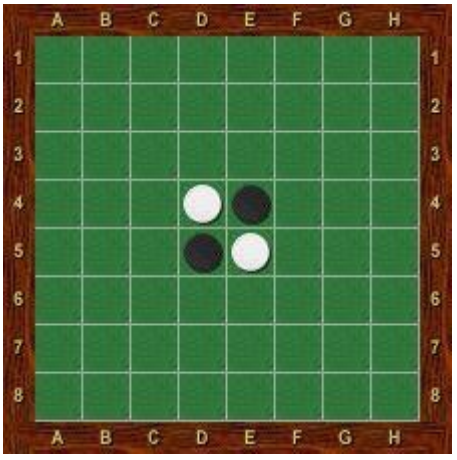
Présentation du projet et objectifs

Le projet a pour but d'implémenter en Python deux jeux de plateau : l'Awele et l'Othello.

L'awele est un jeu de 6x2 cases comptant chacune 4 billes, les joueurs jouent alternativement en déplaçant toutes les billes d'un trou de leur camp et en les égrainant dans toutes les cases qui suivent la rangée de départ en suivant le sens anti-horaire puis la rangée de leur adversaire. Ils gagne des points lorsque la dernière bille tombe dans un trou du camp adverse et qu'il y a deux ou trois billes dans ce trou, les billes récupérés sont ajoutés au score du joueur. S'ajoute alors deux règles, si l'on fait un tour complet en égrainant, on ne remplit pas la case de départ et si l'adversaire n'a plus de billes dans son camp à la fin du coup, on dit qu'il est affamé et le coup est annulé.



L'othello est un jeu de 8x8 cases comptant 2 pions de blancs et noir sur les 4 cases centrales. Les joueurs jouent à tour de rôle en posant un pion jusqu'à ce que les 64 pions soient posés. Un pion est pris s'il est encadré par des pions de la couleur adverse, il est alors retourné et change de couleur. Le gagnant est alors celui qui possède le plus grand nombre de pions de sa couleur.



Une fois l'implémentation effectué, le but du projet est de créer des intelligences artificielles capables de gagner le plus efficacement et le plus souvent ces jeux de société. Cela implique alors de trouver des conditions de victoires (heuristiques) que l'IA doit prioriser et d'améliorer sa recherche de ces dites conditions.

Nous avons pour les deux jeux utilisés différentes heuristiques.

Awele:

Heuristique de Score : L'IA va favoriser les coups augmentant le score du joueur et minimisant celui de l'adversaire.

Heuristique de Mobilité : L'IA va favoriser les coups permettant d'obtenir le plus de possibilités de jeu au prochain tour.

Heuristique de Stabilité : L'IA va favoriser les coups qui installent des situations facilitant la victoire du joueur. Par exemple, pour l'awele, ce sera des situations où l'adversaire possède des cases a 1 ou 2 billes ce qui, pour le tour suivant, augmenterais le score du joueur ; pour l'othello, ce sera des situations où le contrôle du plateau du joueur sera supérieur à celui de l'adversaire comme aux échecs.

Othello:

Heuristique de Score : L'IA va favoriser les coups augmentant le score du joueur et minimisant celui de l'adversaire.

Heuristique de sommets: L'IA calcul un score en fonction des sommets capturé par le joueur et l'adversaire.

Heuristique de Stabilité : À l'aide d'une grille, on stocke pour chaque case du plateau une valeur en fonction en fonction de ses chances d'être retourné. L'IA calcul ensuite un score en fonction de cette grille.

Heuristique de Mobilité (trop coûteux pour l'othello donc non mise en oeuvre): L'IA va favoriser les coups permettant d'obtenir le plus de possibilités de jeu au prochain tour.

Ces différentes heuristiques seront intégrés à une fonction evaluer pour calculer un score qui correspond à l'état de jeu.

Développement des jeux et leur résolution

Afin de connaître le pourcentage de victoire entre deux IA différentes, le main lance une centaine de simulations passant par la fonction unMatch() du fichier game.py qui vérifie dans une boucle que la partie n'a pas remplit la condition de victoire d'un joueur à l'aide de la fonction finJeu() et qui appelle la fonction joueCoup() dans awele.py ou othello.py.

AWELE

Développement : Dans la boucle évoquée précédemment, l'appel à joueCoup() met à jour les différents paramètres de la variable jeu. Tout d'abord le plateau, en appelant distribue() qui distribue les billes en fonction de la case de départ donnée en paramètre à joueCoup() et qui regarde si le joueur n'a pas gagné de points en appelant verifCoupFinal() qui elle même vérifie si le joueur n'est pas affamé avec adversaireAffame(). Une fois le plateau actualisé, la fonction joueCoup() actualise le joueur courant, la liste de ses coups possibles et la liste des coups joués.

OTHELLO

Développement : Dans la boucle évoquée précédemment, l'appel à joueCoup() met à jour les différents paramètres de la variable jeu. Tout d'abord l'ordre des joueurs et l'inverse si l'ordre est déjà établi pour alterner les joueurs. Le plateau vient après à l'aide de la fonction pieceManger() qui teste les différentes directions possibles en regardant si il n'y a pas un pion adverse, allié, un bord ou rien. Les pièces adverses sont mangées, s'il n'y a rien on pose un pion et sinon on ne fait rien. Une fois le plateau actualisé, la fonction joueCoup() actualise la liste des coups possibles, la liste des coups joués et les scores.

Résolution : Othello et Awele comptent quelques IA qui n'ont pas vraiment besoin d'explications que nous allons évoquer ici rapidement.
-Random : IA qui joue aléatoirement créée en important le package random.
-Humain : Une fonction qui demande un input clavier du coup à jouer parmi les coups disponibles au joueur.
-Premier Coup Valide : IA qui joue toujours le premier coup de la liste des coups valides.
-Horizon 1 Efficace : IA qui applique les heuristiques vues précédemment sans méthode particulière.
-Horizon 1 : IA qui utilise seulement l'heuristique de score. (Awele seulement)

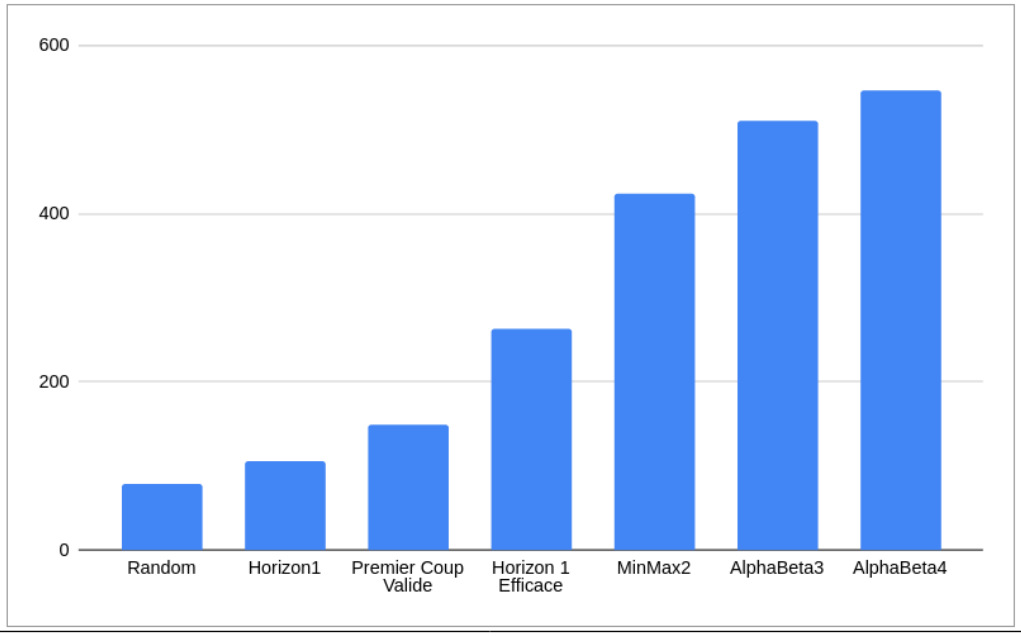
Nous allons maintenant voir les méthode de création d'IA qui étudient le jeu en profondeur, c'est-à-dire qui regarde les différentes possibilités d'évolution du plateau à partir d'un coup donné.
Tout d'abord la méthode MinMax va à une profondeur donnée minimiser le score adverse et maximiser le score heuristique du joueur. La fonction MinMax() va effectuer une boucle de n=profondeur tours en jouant pour le joueur le coup donnant le meilleur score et pour l'adversaire celui donnant le pire à chaque valeur de n.

La méthode Alpha Beta est plus efficace que MinMax en éliminant les coups inutiles car moins forts. AlphaBeta est représentée sous la forme d'un arbre où seul la branche avec le meilleur score Heuristique sera étudié plus en profondeur. Elle va chercher de manière récursive le score le plus grand pour le joueur et le plus faible pour l'adversaire.

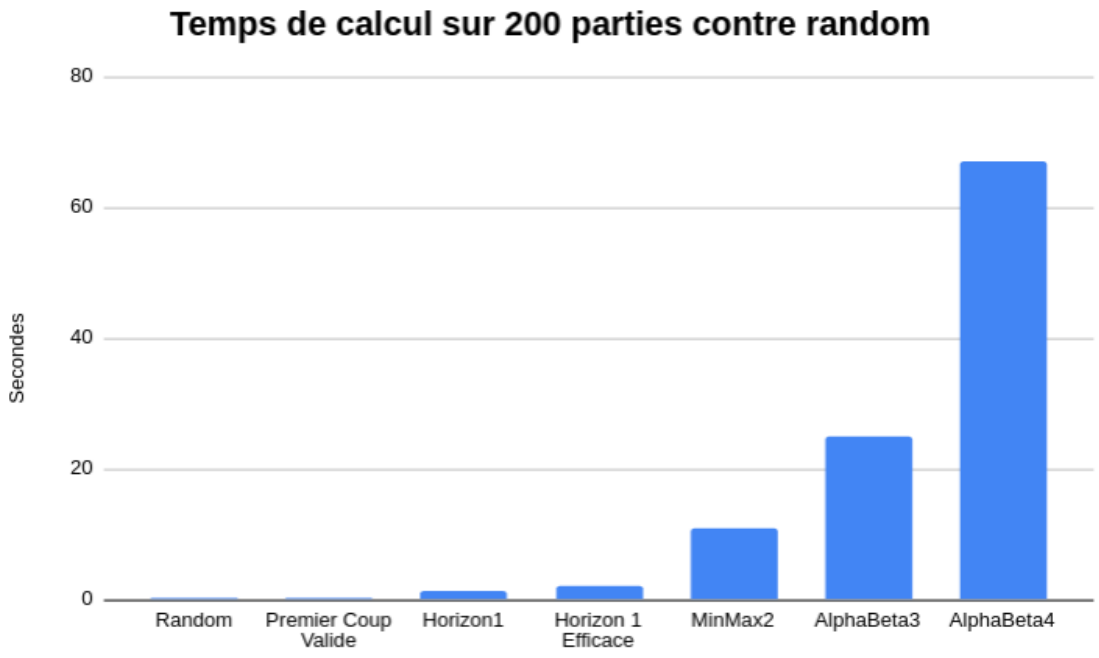
Graphes et Comparaison

Awele

	Random	Horizon1	Premier Coup Valide	Horizon 1 Efficace	MinMax2	AlphaBeta3	AlphaBeta4
Random	_____	59.5%	63%	95%	100%	100%	100%
Horizon1	37.5%	_____	56%	91%	100%	100%	100%
Premier Coup Valide	37%	39%	_____	67%	100%	100%	100%
Horizon 1 Efficace	5%	8%	30%	_____	92%	96.5%	98%
MinMax2	0%	0%	0%	5%	_____	84%	80%
AlphaBeta3	0%	0%	0%	3%	15%	_____	68%
AlphaBeta4	0%	0%	0%	1.5%	17%	29.5%	_____
Somme	**79.5**	**106.5**	**149**	**262.5**	**424**	**510**	**546**

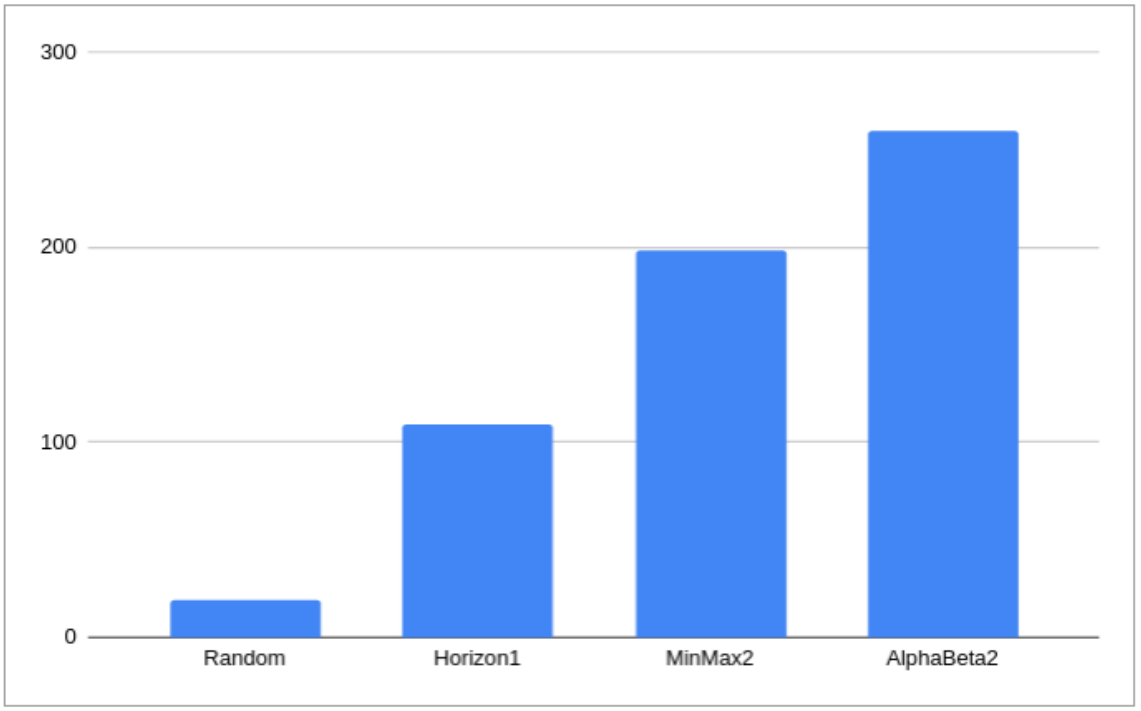


Pourcentage de victoire sur 200 parties

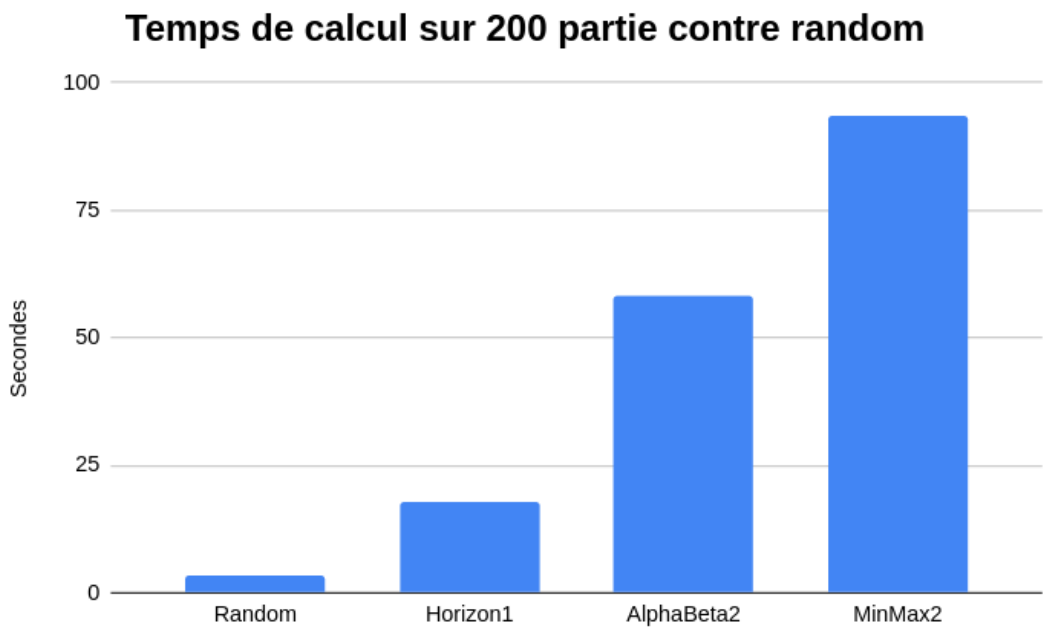


Othello

	Random	Horizon1	MinMax2	AlphaBeta2
Random		88%	92%	94.5%
Horizon1	8.5%		83%	90.5%
MinMax2	6.5%	12%		74%
AlphaBeta2	4.5%	9%	23%	
Somme	**19.5**	**109**	**198**	**259**



Pourcentage de victoire sur 200 parties



Conclusion

- Plus on utilise d'heuristiques pour le calcul du score, mieux le winrate est.

- Plus la profondeur est élevée, plus on calcul de coup en avance, donc plus on gagne.

- Plus la profondeur est élevée, plus il y’a de calcul à effectuer, donc plus le temps de calcul est élevé.

- Pour une même profondeur l’algorithme alpha-beta est plus efficace que minmax, donc alpha-beta est plus rapide.

Algorithmes d’apprentissages

Initialisation

Initialisation le tableau des poids de chaque heuristic: params
Initialisation du pas d’apprentissage: alpha
Initialisation de l’indice de convergence: indConvergence
Choisir le joueur oracle auquel on veut s’approcher

Boucle

Tant qu’il n’y a pas convergence: abs(params - paramsPrec) < indConvergence
    Simuler une partie de jeu contre un joueur aléatoire
        Pour chaque coup valide:
        Récupérer pour l’oracle le score
            Récupérer pour le joueur paramétrique le score
            Mettre à jour les poids du joueur paramétriques
                Pour chaque coup de l’oracle autre que le coup optimal
                    o = somme(params * meilleur score joueur paramétrique)
                    s = somme(params * score coup du joueur paramétrique)
                    Si (o - s) < 1:
                        Pour chaque poid(param)
                            scjC = score du joueur pour le coup
                            scjOpt = score du joueur pour le coup optimal oracle
                            pramam = param - alpha * (scjC - scjOpt)
            Faire jouer le joueur paramétriques
            Faire jouer l’adversaire
    Faire baisser légèrement le pas d’apprentissage: alpha

Du fait d’une trop grande variation entre chacun des scores obtenus les poids obtenus en sorties de boucle converge très vite vers l'infini ou très grand, les données sont donc inutilisables.