Projet: Othello et Awele (IA)

21.05.2022

—

Tawbi Chadi et Chaou Rayan
Numéro etudiant: 28706563 / 28706331
2ème année Licence Informatique
Mono-disciplinaire Groupe n°1 2021-2022

SORBONNE UNIVERSITÉ

Sommaire

Introduction

Ce projet consiste à la création de 2 jeux de plateaux à 2 joueurs, Othello et Awalé. En se basant sur le langage de programmation de Python, nous allons donc créer les éléments essentiels à la gestion du plateau, tout en réfléchissant au côté intelligence artificielle (mise en place d'algorithmes à apprentissage supervisé, par exemple) représenté par les différents joueurs.

Notre but est de pouvoir comparer chaque joueur et d'en déduire le plus intelligent et optimal.

Développement des Jeux

Pour commencer, on à un fichier **game.py** générique qui permettra d'avoir les fonctions nécessaires relatives aux 2 jeux, ce sont des fonctions communes qui sont donc utilisées par les 2 jeux, d'où la généricité. Cela permet d'avoir un code plus ergonomique, robuste et plus facile à lire.

Les différentes fonctions, ainsi que leur utilisation sont définies dans le fichier **game.py.**

getCopieJeu(jeu)	clearArea()
finJeu(jeu)	getPlateau(jeu)
saisieCoup(jeu)	getCoupsJoues(jeu)
getCoupsValides(jeu)	getScores(jeu)
coupValide(jeu,coup)	getJoueur(jeu)
joueCoup(jeu,coup)	changeJoueur(jeu)
initialiseJeu()	getScore(jeu,joueur)
affiche(jeu)	getCaseVal(jeu, ligne, colonne)
generatePrint(length, beginLength)	updateScores(jeu, scoreToAdd, joueur)

Les fonctions définies dans le fichier **game.py** utilisent une liste **jeu**, qui suit le format suivant:

[

Plateau (Matrice x*y ou x représente le nombres de lignes et y représente le nombres de colonnes),

Joueur(Int = 1 ou 2),

Liste des Coups Valides (Liste de Tuple (int,int)),

Liste des coups joués (Liste de Tuple (int,int)),

Score (Tuple (a,b))

]

*Cette liste est initialisé en fonction du jeu via la fonction **game.initialiseJeu()***

Grâce au fichier game.py, on peut maintenant coder les 2 jeux séparément on créé 2 nouveaux fichiers qui se nomment respectivement **othello.py** et **awele.py.**

Dans ces fichiers sont définies des fonctions spécifiques pour chacun des jeux:

- Des fonctions qui ont pour but de démarrer le jeu (**initialiseJeu()**)
- Des fonctions permettant de faire jouer un joueur (**joueCoup(jeu, coup), listeCoupsValides(jeu)**)

  Pour Othello:

  - estSurPlateau(x, y)
  - chercheCoupsValide(plateau, tile, xstart, ystart)
  - getPlayerTile(jeu)

  Pour Awele:

  - getNextCase(case, antiHoraire=True)
  - estAffame(jeu, joueur)
  - estValide(jeu, coup, checkNourrit=False)
- Une fonction permettant de savoir si le jeu touche à sa fin (**finJeu(jeu)**)

2 Dossiers ont été créés pour faciliter la lecture du code entre les 2 jeux:

- Un dossier Othello pour le jeu othello accompagné de son **main.py** et ses différents Joueurs dans le dossier **./Othello/Joueurs**
- Un dossier Awele pour le jeu awale accompagné de son **main.py** et ses différents Joueurs dans le dossier **./Awele/Joueurs**

Conception des Joueurs

Joueur Humain:

Le joueur humain est simplement l'affichage du plateau, avec l'affichage de la liste des coups valide. Après l'affichage, on demande à l'utilisateur de choisir un coup parmi la liste des coups valides. Si la liste de coups valides est vide alors le jeu s'arrête, sinon on joue le coup sélectionné sans affamer l'adversaire (pour le cas Awele). On a donc deux jeux fonctionnels entre deux humains.

Joueur Aléatoire:

Nous avons aussi implémenté un joueur aléatoire qui joue aléatoirement un coup valide parmi la liste des coups valides. Celui-ci a pour but d'être facilement battus par nos algorithmes.

Les joueurs humains et Aléatoire permettront d'affronter les prochains joueurs conçus avec une IA.

Joueurs/Algorithmes Min-Max et Alpha-beta:

Ces algorithmes ont pour but de parcourir des arbres de possibilités à l'aide de plusieurs fonctions avec une variable **prof** (variable qui correspond à la profondeur de l'arbre) qui limite le nombre de niveaux explorés pour éviter des surcharges de mémoire, des plantages du programmes ou encore des "timeout".

Ces 2 algorithmes, avec plusieurs similitudes se base sur 3 fonctions principales:

- **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 prend en compte un coup et parcourt l'arbre jusqu'à la profondeur indiquée dans le 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 prend la liste des coups valide et fait appel à estimation pour chacun d'entre eux pour décider du meilleur coup possible.

L'algorithme **Mini-max** (**joueur_minmax.py**) va parcourir tous les coups possible à un moment précis du jeu (avant qu'un joueur joue) et pour chaque coup possible, l'algorithme va déterminer toutes les parties qui peuvent en découler. Lorsque l'on explore un nœud, l'algorithme va déterminer s'il est en train d'analyser son propre coup (**max**) ou si ce coup correspond au coup que l'adversaire va jouer (**min**). S'il est en **max**, il doit choisir le coup qui lui permettra d'obtenir maximum de point donc de remporter la partie mais s'il est en **min** il doit choisir le coup qui lui rapportera le minimum de points, tout en se plaçant dans la situation de l'adversaire.

L'algorithme **Alpha-bêta (joueur_alphabeta.py)** est une version améliorée de l'algorithme **Mini-Max**. Si un coup donné implique que le coup du noeud **Min** joué soit peu avantageux alors l'algorithme n'en tiendra pas compte, cela permet donc de réduire le nombre de noeuds de l'arbre découverts et marqués par l'algorithme, et de déterminer le meilleur coup avec une meilleur profondeur

Joueurs à Apprentissage / Apprentissage Supervisé:

Après nos tests des algorithmes **Mini-Max** et **Alpha-Beta**, nous avons aperçu un fort ralentissement du 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. Celui-ci ressemble à l'algorithme **Alpha-Beta**, sauf pour la fonction évaluation qui sera modifiée.

La fonction **évaluation** 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'entraînement.

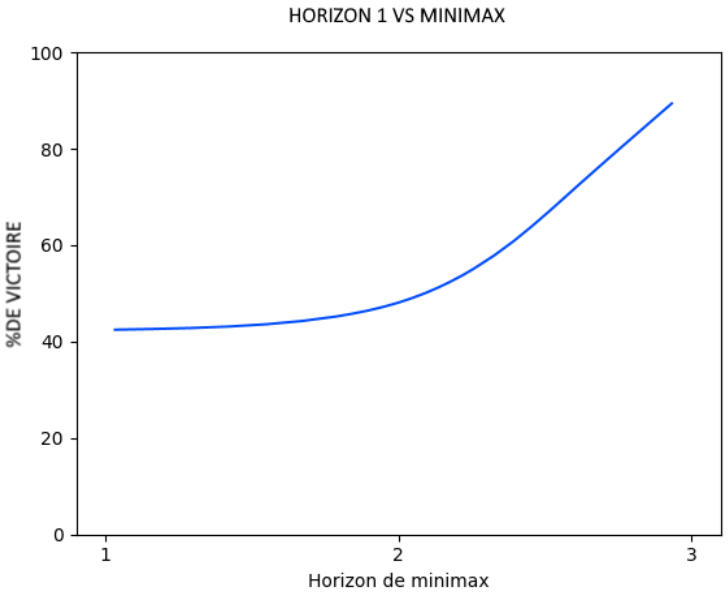
On à donc:

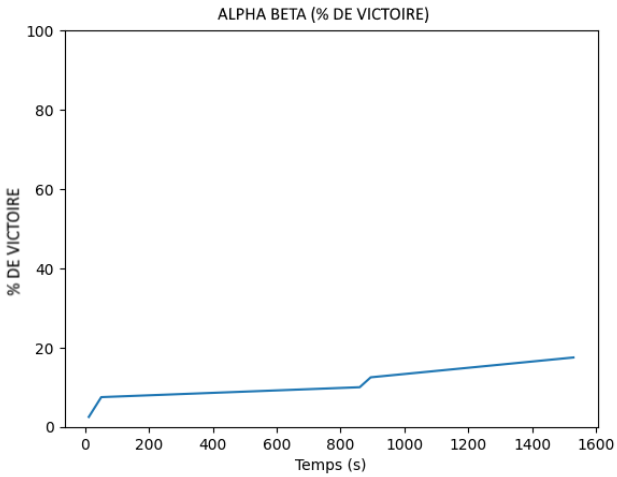
- **Un joueur adversaire**, qui ne sert pas vraiment à grand chose à part faire varier les parties
- **Un oracle** (ou enseignant) qui aura pour but de jouer la partie en même temps que le joueur supervisé - Le joueur intelligent que l'on souhaite améliorer De chaque partie il faudra réajuster les différents coefficients a* de l'élève et réessayer

Courbes de résultats et analyse

Awele

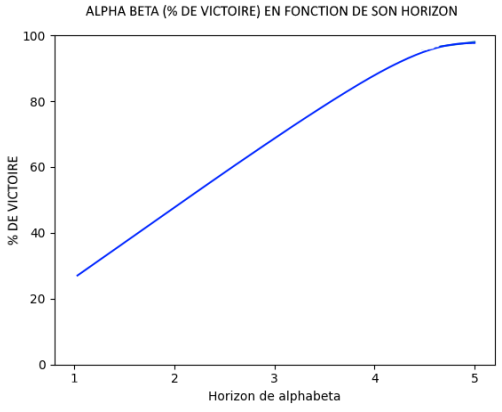
Le 1er graphique représente le pourcentage de victoire de minimax contre l'algorithme d'horizon 1 en fonction de l'horizon de mini-max.
On voit que plus l'horizon augmente
Plus le pourcentage de victoire de minimax augmente. Ce qui montre que minimax est plus performant.





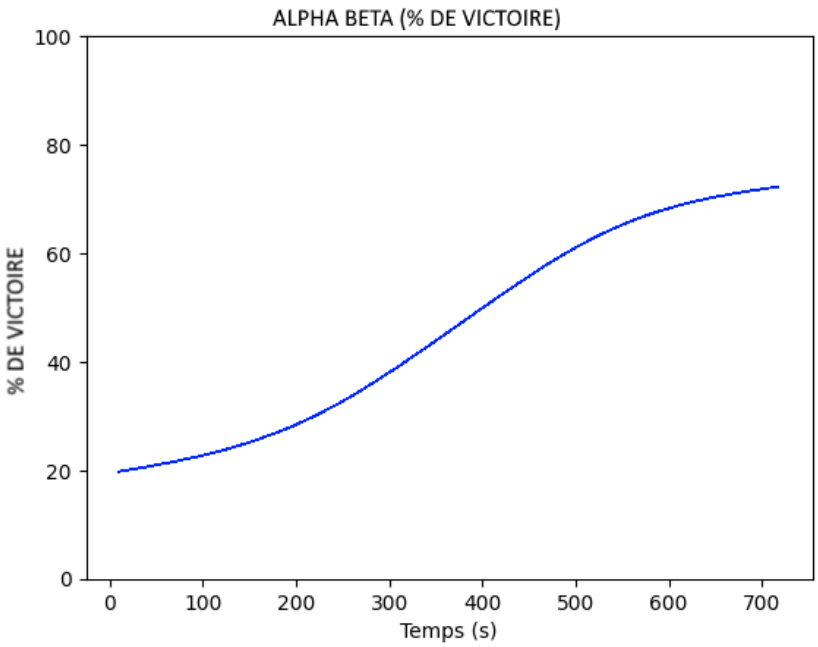
Le 2ème graphique représente le pourcentage de victoire d'alpha beta fonction du temps en secondes de mini-max.
On voit que plus le temps augmente
Plus le pourcentage de victoire d'alpha beta augmente. Ce qui montre qu'alpha beta est plus performant dans le temps.

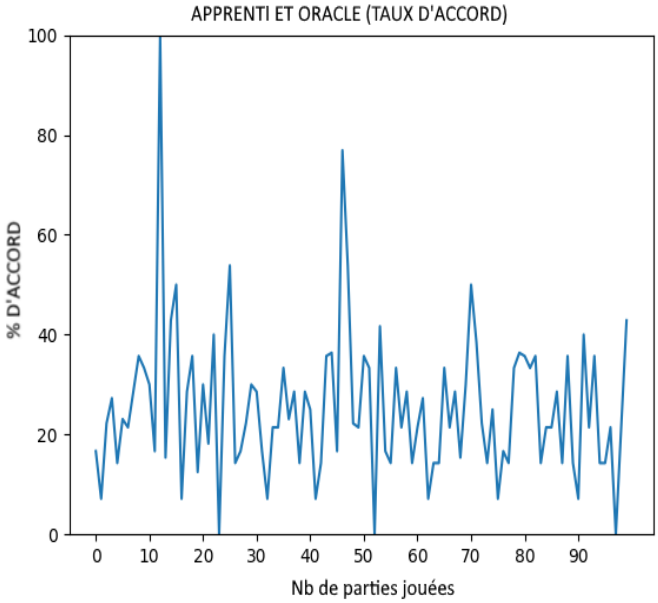
Le 3ème graphique représente le pourcentage de victoire d'alpha beta en fonction de son horizon. On voit que plus l'horizon est grand meilleur est son taux de victoire face à un joueur horizon 1. On remarque alors qu'alpha-beta est meilleur que l'algorithme mini-max.



Othello

Le 1er graphique représente le pourcentage de victoire d'Alpha-Beta contre l'algorithme d'horizon 1 en fonction du temps en secondes.
On voit que plus le temps augmente plus le pourcentage de victoire d'Alpha-Beta augmente. Ce qui montre qu'Alpha-Beta est plus performant dans le temps.





Le 2ème graphique représente le

Taux d'accord entre le joueur apprenti et le joueur oracle. On remarque que malgré un nombre important de parties. Les 2 joueurs ont du mal à se mettre en relation… (Moyenne de 20% d'accord entre les 2 joueurs)

Cela peut être dû au manque d'entraînement du joueur apprenti, avec des dizaines de milliers de parties, le joueur apprenti serait en accord avec Oracle.