

TP1, IFT3335

Jeu d'Othello

Date de remise

À rendre avant le 10 mars 2025, 23:59

A faire en groupe de 2 personnes. Exceptionnellement, avec autorisation, il est possible de faire ce TP seul ou en groupe de 3 personnes.

Le jeu

Ce TP vise à pratiquer la recherche heuristique dans un jeu à deux joueurs. Il consiste à la réalisation de différents algorithmes pour jouer le jeu d'Othello. La règle du jeu est décrite sur le site suivant : <https://www.ffothello.org/othello/>. Ce site décrit aussi un ensemble d'algorithmes typiques applicables au problème, dont minimax, alpha-beta et leurs variantes : <https://www.ffothello.org/informatique/algorithmes/>.

En bref, un joueur possède des pièces noires et l'autre des pièces blanches, et chacun joue à tour de rôle. En plaçant une pièce à une case A, si cette case est dans une ligne horizontale, verticale ou diagonale avec une autre pièce B du même joueur, séparé par des pièces de l'adversaire, alors les pièces de l'adversaire entre A et B sont retournées (flip). Dans cette opération de *flipping*, il ne devrait pas avoir de case vide entre A et B. Un joueur peut placer une pièce seulement s'il peut retourner des pièces adversaires. Sinon, il passe son tour. Le jeu s'arrête quand aucun joueur ne peut placer une pièce. Le joueur ayant le plus grand nombre de pièces gagne le jeu.

La configuration de départ est toujours comme la figure à gauche, où O représente une pièce blanche et X une pièce noire. C'est le joueur Noir qui commence. En plaçant une pièce à la position (2, 3), la configuration sera transformée en celle de la droite (note : la position en haut à gauche est (0, 0)).

			O	X			
			X	O			

			X				
			X	X			
			X	O			

Les algorithmes possibles

Pour ce jeu, il est possible d'utiliser différents algorithmes:


1. Stratégie avec minimax : Le joueur utilise l'algorithme minimax pour choisir une position à placer une pièce. Une implémentation de base de cet algorithme est fournie sur la plateforme.
2. Stratégie avec alpha-beta : Le joueur utilise l'algorithme alpha-beta pour choisir la position.

3. Stratégie avec la recherche arborescente Monte Carlo : Le joueur utilise une exploration avec la recherche arborescente Monte Carlo pour choisir une position.

L'algorithme minimax est déjà partiellement implémenté avec une fonction d'évaluation simple – voir le code source de la plateforme. On vous demande d'implémenter et tester les autres algorithmes (voir les tâches).

Plateforme disponible

Pour faciliter votre implémentation, une plateforme est offerte, vous permettant de visualiser le jeu, faire jouer votre stratégie choisie contre un autre joueur (minimax de base avec 4 plis), et faire un match entre deux algorithmes. La plateforme fournit le contrôle du jeu, permettant à chaque joueur de jouer à tour de rôle. Le lien de l'application est : <https://othelloift3335.streamlit.app/>. En cliquant

sur  de la page de la plateforme, vous pouvez voir le code du programme.

Cette plateforme est illustrée comme suit :

Othello - Compétition TP1 ift3335 !

Compétition entre IA !

Entrez votre ID étudiant

0000

Entrez le nom de votre équipe

NameExample

Soumettez votre propre IA sous forme de fonction Python.

Entrez votre code Python ici :

```
def user_ai(board, player):
```



Votre IA a été enregistrée pour l'équipe NameExample et 0000 id!

Lancer la partie IA vs Minimax AI

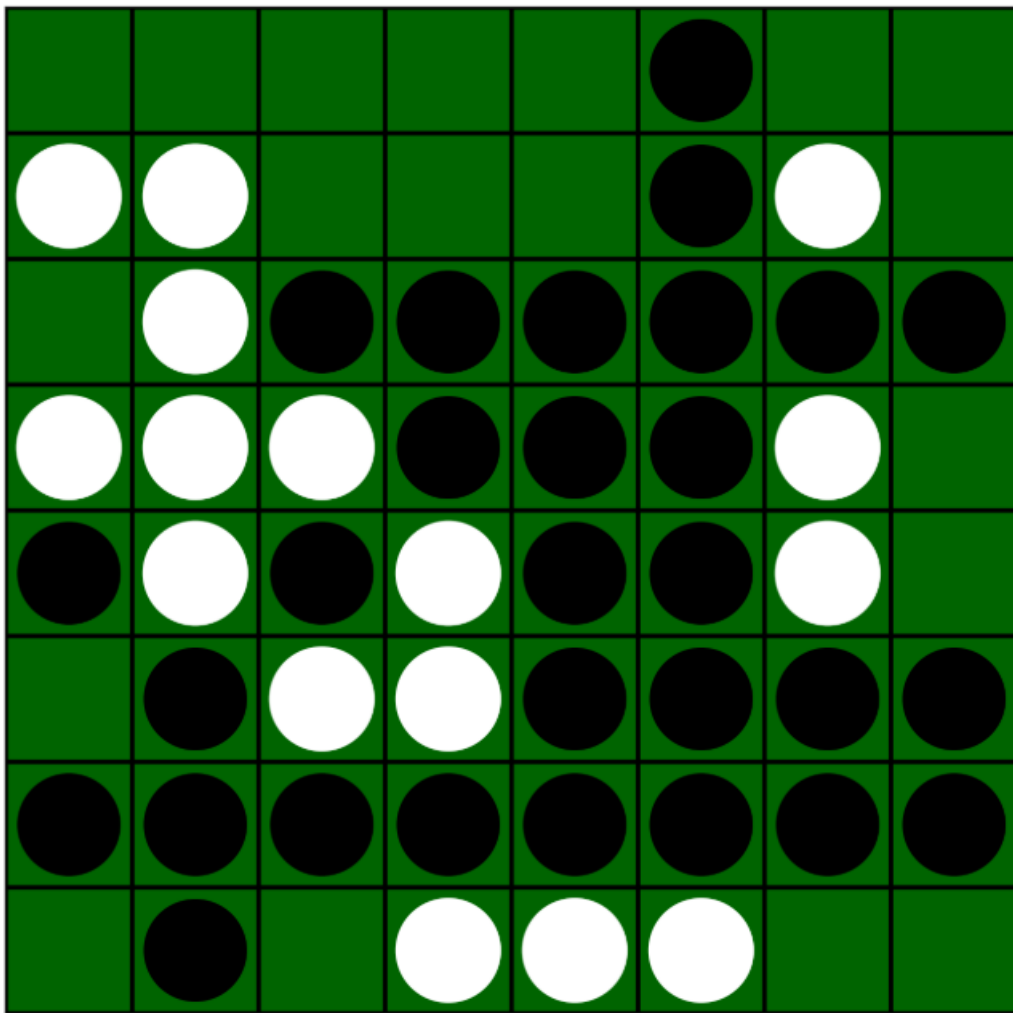
Classement des étudiants

	ID	Nom d'équipe	Score
--	----	--------------	-------

Pour faire jouer votre algorithme contre le joueur de base, vous devez copier votre implémentation de la fonction **def user_ai(board, player)** dans la fenêtre en haut. Quand vous lancer le jeu, votre algorithme sera le joueur Noir qui commence, et à tour de rôle, c'est votre algorithme ou l'algorithme de base qui sera demandé à retourner le prochain pas (move). La configuration sera affichée au fur et à mesure.

Votre IA a été enregistrée pour l'équipe NameExample et 0000 id!

Lancer la partie IA vs Minimax AI



Dans la deuxième partie de la plateforme, on offre la possibilité de faire jouer deux algorithmes. Pour cela, il faut copier les deux algorithmes que vous avez implémentés dans deux fenêtres différentes. Ces joueurs seront nommés **user_ai1** et **user_ai2**. Ces deux joueurs vont faire un match jusqu'au bout.

En plus de l'interface, la plateforme vous offre aussi quelques implémentations de fonctions utiles :

1. **Test d'une position légale is_valid_move():** Tester si une position est légale : pour un joueur, il peut seulement placer une pièce dans une case où il est possible de flipper les pièces de l'adversaire.
2. **Liste des positions légales get_valid_moves():** Déterminer l'ensemble de positions légales : En utilisant la fonction précédente, on détermine toutes les positions légales pour le joueur.
3. **Joueur minimax de base minimax():** Ce joueur utilise minimax avec 4 plis pour choisir la position à jouer. La fonction d'évaluation pour les nœuds terminaux est la différence entre le nombre de pièces du joueur et celui de son adversaire dans la configuration.

Tâche 1 : Minimax amélioré

Implémenter une version de minimax avec un nombre de plis plus élevé : 6 plis, et avec une fonction d'évaluation de nœuds terminaux améliorée.

En plus de la différence entre les pièces de deux joueurs déjà implémentée dans minimax de base (critère 1), la fonction d'évaluation améliorée tient aussi compte de la valeur de chaque case (critère 2 – la valeur de case), ainsi que le nombre de positions que le joueur peut jouer (critère 3 - la mobilité). La valeur d'une case représente son potentiel dans le jeu. Typiquement, une pièce place dans un coin ne pourra pas être flippée par l'adversaire et peut servir comme un pont pour des flips ultérieurs. Ainsi, placer une pièce dans un coin représente une grande valeur potentiel pour le joueur. Les valeurs peuvent prendre la forme comme dans l'illustration suivante :

500	-150	30	10	10	30	-150	500
-150	-250	0	0	0	0	-250	-150
30	0	1	2	2	1	0	30
10	0	2	16	16	2	0	10
10	0	2	16	16	2	0	10
30	0	1	2	2	1	0	30
-150	-250	0	0	0	0	-250	-150
500	-150	30	10	10	30	-150	500

Le critère de mobilité permet de savoir si le joueur a beaucoup d'alternatives dans une configuration. On préfère une configuration qui offre plus de positions à jouer au joueur.

Dans votre implémentation, vous pouvez combiner ces trois critères d'une certaine façon (e.g. combinaison linéaire), et tester sa performance.

Vous êtes encouragés à intégrer d'autres heuristiques dans votre fonction d'évaluation. Cherchez sur le web les heuristiques trouvées par des experts, et implémentez celles qui sont à votre portée.

Tâche 2 : Algorithme alpha-beta

Utiliser l'algorithme alpha-beta à la place de minimax. Grâce aux élagages, certaines branches ne seront pas explorées. On peut donc explorer plus profondément. On suppose que l'économie générée avec les élagages nous permet d'explorer un pli plus profond, c'est-à-dire à 7 plis.

Faire jouer cet algorithme alpha-beta contre minimax et faites votre analyse sur ces deux algorithmes dans votre rapport.

Tâche 3 : Utiliser la recherche arborescente Monte Carlo

Pour ce joueur, au lieu d'utiliser une fonction heuristique pour choisir une position à jouer, il utilise la recherche arborescente Monte Carlo pour le faire. Comme présenté dans la classe, cet algorithme explore les enfants de la configuration courante en privilégiant l'enfant qui a la meilleure valeur de UCB1 et continue à jouer une configuration peu explorée avec des jeux au hasard jusqu'au bout pour obtenir du feedback. Cet algorithme n'utilise pas d'heuristique définie par des experts, et peut potentiellement trouver une meilleure stratégie qu'un expert humain.

Plutôt que de déterminer un nombre d'explorations variable selon le temps disponible, on fixe ce nombre à 10 000 pour chaque tour dans ce TP. Vous pouvez aussi faire varier ce nombre dans votre analyse.

Faites jouer ce joueur contre les algorithmes que vous avez programmés en haut.

À rendre

Vous devez soumettre votre implémentation des algorithmes demandés sur Studium. Vous pouvez utiliser des parties des implémentations fournies dans la plateforme quand c'est utile, par exemple, l'affichage, l'implémentation minimax de base, le contrôle du jeu, et les outils tels que la fonction qui teste si une position est légale, le changement de la configuration par suite d'un pas (move) d'un joueur.

En plus de votre programme, vous devez aussi remettre un rapport d'au plus 4 pages pour décrire brièvement les algorithmes et les heuristiques que vous avez implémentés, faire des analyses sur les algorithmes (comparaison de performance, du temps d'exécution, et d'autres discussions).

Chaque équipe est demandée à faire une remise. Indiquez les noms de tous les partenaires dans l'équipe dans votre remise. La date limite pour la remise est le 10 mars, 23:59. Chaque jour de retard sera pénalisé de 20% dans la note.

Barème de correction

Ce TP compte pour 15 points dans la note finale. Cette partie de notes sera attribuée selon les proportions suivantes :

- Implémentation de minimax amélioré : 25%
- Implémentation de alpha-beta : 25%
- Implémentation de recherche arborescente Monte Carlo : 25%
- Le rapport décrivant les heuristiques que vous implémentez et votre analyse sur les algorithmes : 25%
- Bonus : En fonction des nouvelles heuristiques ajoutées dans votre implémentation, jusqu'à 10% de bonus peut être accordé.

Tournoi (facultatif)

Vous êtes encouragés à améliorer votre algorithme. Pour ce faire, vous pouvez soumettre votre algorithme à la plateforme pour jouer contre le joueur minimax de base, en utilisant un nom de l'équipe. Le score est la différence en nombre de pièces entre votre joueur et le joueur minimax de base. Votre meilleur score sera mis dans le tableau de classement. Pour être juste, on impose une limite de 6 plis pour minimax et 7 pour alpha-beta s'ils sont utilisés dans cette compétition. Pour

l'algorithme avec la recherche arborescente Monte Carlo, le nombre d'exploration ne doit pas dépasser 100 000. Vous pouvez tester ces algorithmes avec plus de plis et plus d'explorations vous-mêmes localement (sans soumettre sur la plateforme).

Les 8 équipes qui ont les meilleurs scores dans le classement seront invitées à faire un tournoi pendant une séance de démonstration : ces équipes seront regroupées en 4 pools de façon aléatoire, et le gagnant de chaque pool ira en semi final. Les 4 équipes retenues vont être appariées au hasard en 2 groupes. Le gagnant de chaque groupe ira en final. L'équipe qui gagne le match final sera le champion du tournoi.

Comme prix, le champion recevra un bonus de 10% additionnel, et l'équipe en seconde place un bonus de 5% additionnel.