Projet programmation L2 - Othello
2022/2023
Lamia Benamara

L'objectif principal de ce projet est de développer vos compétences en programmation et de mettre en pratique les différentes notions théoriques acquises en programmation orienté objet ainsi qu'en algorithmique et structures de données.

Le projet consiste à implémenter différentes versions du jeu Othello : la version basique permettant d'affronter deux joueurs humains et la version permettant à un joueur humain d'affronter un ordinateur (IA). La qualité de l'IA est déterminée en fonction de la stratégie (algorithme) adoptée. Le cahier des charges est rédigé par niveau de difficulté. Vous devez suivre l'ordre des questions afin de progresser efficacement.

Présentation :

Othello est un jeu de société combinatoire abstrait, qui oppose deux joueurs : Noir et Blanc.
Il se joue sur un tablier unicolore de 64 cases, 8 sur 8, appelé othellier. Les colonnes sont numérotées de gauche à droite par les lettres a à h et les lignes sont numérotées de haut en bas par les chiffres 1 à 8. Les joueurs disposent de 64 pions bicolores, noirs d'un côté et blancs de l'autre. En début de partie, quatre pions sont déjà placés au centre de l'othellier : deux noirs, en e4 et d5, et deux blancs, en d4 et e5.
Chaque joueur, noir et blanc, pose l'un après l'autre un pion de sa couleur sur l'othellier selon les règles définies ci-dessous. Le jeu s'arrête quand les deux joueurs ne peuvent plus poser de pion. On compte alors le nombre de pions. Le joueur ayant le plus grand nombre de pions de sa couleur sur l'othellier a gagné.

Source : Wikipedia

Bien que semblable au jeu de GO, l'Othello est plus facile à étudier en matière d'intelligence artificielle car l'arbre de recherche est relativement petit (faible profondeur).

Règles du jeu :

Au début de la partie, deux pions noirs sont placés en e4 et d5 et deux pions blancs sont placés en d4 et e5.

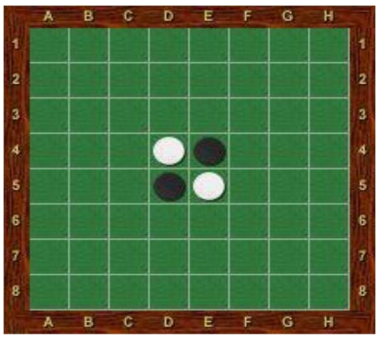

Figure 1 – Situation de départ

C'est le joueur possédant les pions noirs qui commence et les deux adversaires jouent ensuite à tour de rôle.

Comment poser un pion ?
A son tour de jeu, le joueur doit poser un pion de sa couleur sur une case vide de l'othellier, adjacente à un pion adverse. Il doit également, en posant son pion, encadrer un ou plusieurs pions adverses entre le pion qu'il pose et un pion à sa couleur, déjà placé sur l'othellier. Il retourne alors de sa couleur le ou les pions qu'il vient d'encadrer. Les pions ne sont ni retirés de l'othellier, ni déplacés d'une case à l'autre. Le premier coup de Noir est, par exemple, en f5.

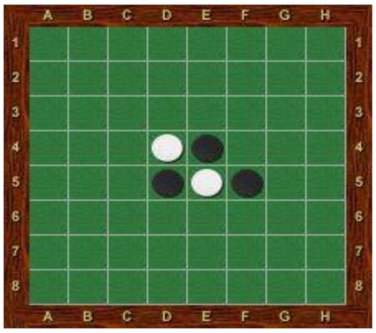

Figure 2 – Premier coup Noir

En jouant f5, il encadre le pion blanc e5 entre le pion qu'il pose et un pion noir déjà présent (ici d5). Il retourne alors ce pion.

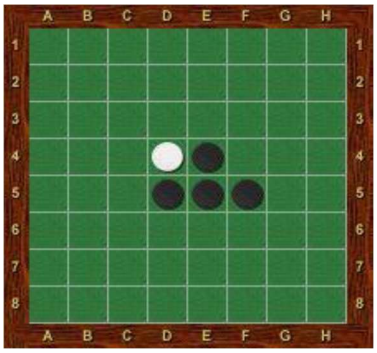

Figure 3 – Retournement du pion blanc avec le coup f5

Le joueur Noir aurait aussi pu jouer en e6, c4 ou d3. Notons que ces quatre coups de Noir sont parfaitement symétriques. Noir n'a donc pas à réfléchir pour choisir son premier coup.
C'est maintenant au joueur Blanc de jouer. Il a trois coups possibles. En effet, il est obligatoire de retourner au moins un pion adverse à chaque coup. Blanc peut donc jouer en f4, f6 ou en d6.

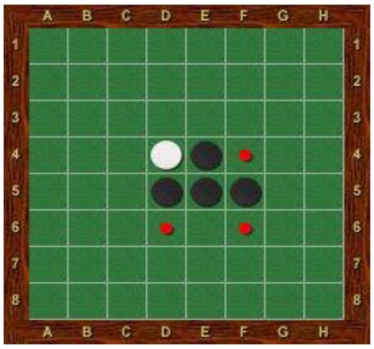

Figure 4 – Possibilités du coup Blanc

On peut encadrer des pions adverses dans les huit directions. Par ailleurs, dans chaque direction, plusieurs pions peuvent être encadrés. On doit alors tous les retourner.

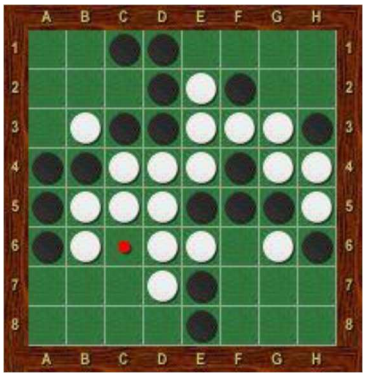
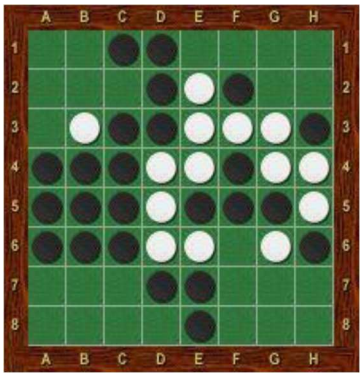

Figure 5 – Noir joue en c6 / Figure 6 – Résultat après c6

Le joueur Noir a joué en c6. Il retourne alors les pions b6 (encadré grâce à a6), b5 (encadré grâce à a4), d7 (encadré grâce à e8), c5 et c4 (encadrés grâce à c3). Notons que ni d6, ni e6 ne sont retournés à cause de la case vide en f6. Il n'y a pas de réaction en chaîne : les pions retournés ne peuvent pas servir à en retourner d'autres lors du même tour de jeu.

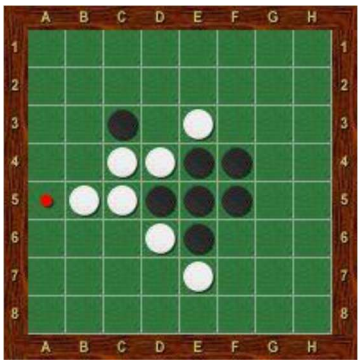
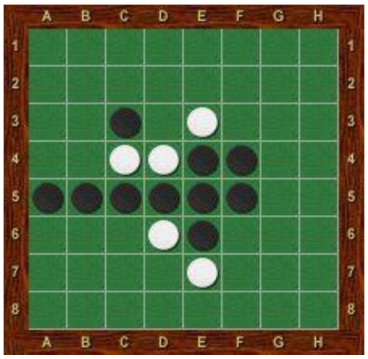

Figure 7 – Noir joue en a5 / Figure 8 – Résultat après a5 (c4 reste Blanc)

La partie est terminée lorsqu'aucun des deux joueurs ne peut plus jouer. Cela arrive généralement lorsque les 64 cases sont occupées. Mais il se peut qu'il reste des cases vides où personne ne peut plus jouer : par exemple lorsque tous les pions deviennent d'une même couleur après un retournement ou bien comme sur cette position.

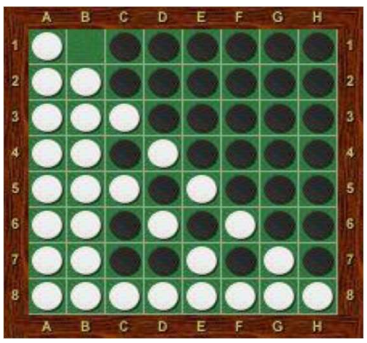

Figure 9 – Fin de partie

Aucun des deux joueurs ne peut jouer en b1 puisque aucun retournement n'est possible. On compte les pions pour déterminer le score. Les cases vides sont données au vainqueur. Ici, Blanc a 29 pions, Noir 34 et il y a une case vide. Donc Noir gagne par 35 à 29.

Source : http://www.ffothello.org/othello/regles-du-jeu/

I. Partie 1 : Jeu de base (humain vs humain)

Dans cette première partie, vous devez implémenter une version du jeu Othello en mode console où s'affrontent deux joueurs humains. Le mode console se traduit par un affichage, avec des caractères, de la grille et son actualisation après chaque coup.
Vous devez réfléchir à votre conception et au choix de vos classes.

Implémenter le jeu de base en suivant les instructions suivantes :

- L'Othellier est par défaut une matrice de 8x8 cases (on peut changer la taille)
- Possibilité de sauvegarder une partie et de la charger
- Les coups doivent être légitimes (vérification de tous les cas possibles)
- Score en fin de partie
- Plateau représenté en console avec des couleurs
- De préférence pas de saisie des coups, mais un déplacement à l'aide du clavier (utilisez les touches *z, q, s, d*) sur le damier et on joue son coup en appuyant sur la touche *Entrée*.

II. Partie 2 : L'intelligence Artificielle (humain vs IA)

Un peu de vocabulaire qui sera valable pour l'ensemble du projet :

- **Nœud** : Correspond à un état donné du jeu à un instant *t*. Dans le cas de l'Othello, il s'agit de la position des pions
- **Transition** : Action permettant de passer d'un état à un autre. Dans le cas de l'Othello, il s'agit d'un coup (Noir ou Blanc)
- **Arbre de recherche** : Ensemble des états accessibles en partant de l'état actuel. Dans le jeu de l'Othello, la création d'un nouvel état se fait lorsque l'on pose un pion.

A. IA naïve :

Ce niveau est accessible pour tout le monde. Vous devez :

- Implémenter un premier algorithme naïf permettant à un joueur de jouer contre l'ordinateur. Le premier algorithme naïf possible est de que l'ordinateur jouera aléatoirement, vous pourrez proposer une stratégie donnant de meilleurs résultats que l'aléatoire en justifiant vos choix.
- Modéliser les différents coups possibles l'ordinateur par un arbre de recherche qui sera affiché (de la meilleur façon possible) sur la console.

B. IA basée sur MinMax :

Vous verrez que l'algorithme précédent est loin d'être satisfaisant. En effet, l'IA n'a aucune stratégie et n'a aucun moyen d'évaluer un coup donné pour savoir si elle est entrain de gagner ou de perdre.

L'algorithme MinMax est un algorithme adapté aux jeux dits à *somme nulle* (le gain d'un joueur est la perte de l'adversaire) et à information complète (lors du choix d'un coup, chaque joueur connaît

précisément les différents coups possibles ainsi que les coups possibles de son adversaire). Parmi ces jeux, on trouve l'Othello, mais également le Morpion, le Puissance 4, Go, … etc.

Il s'agit d'un algorithme basé sur le théorème de Minimax de Von Neumann. C'est un algorithme efficace mais assez coûteux en temps et en espace. Il existe plusieurs améliorations et son amélioration la plus connue est l'algorithme « MinMax avec élagage Alpha-Beta ».

L'algorithme de base MinMax complet ou exhaustif correspond au cas idéal (mais difficilement réalisable) qui consiste à explorer tout l'arbre de jeux. Le but de MinMax est de trouver le meilleur coup à jouer à partir d'un état donné du jeu lui permettant de gagner.

Le MinMax complet peut éventuellement être implémenté sur des jeux de réflexion très simples tels que le morpion, avec une petite grille de 3x3. Mais pour des jeux plus complexes et avec une grille plus grande comme l'othello, la profondeur et la largeur de l'arbre généré risque de devenir trop grande, il n'est pas toujours possible ou judicieux de le générer intégralement (car espace mémoire ou durée trop grande).

Dans cette partie, vous devez donc mettre en œuvre les deux variantes de MinMax suivantes utilisant une profondeur d'arbre prédéfinie:

1. **Niveau 1 : MinMax à profondeur donnée**

   Le principe de la génération de l'arbre est le même que le MinMax complet sauf qu'on arrête la génération à une profondeur donnée fixée au début.

   Cela signifie que le critère d'arrêt de la génération de l'arbre de jeu est l'atteinte d'une hauteur maximale. Les feuilles de l'arbre ne sont donc pas forcément des configurations de fin de jeu et il n'est généralement pas possible de dire si une feuille correspond à une coup gagnant ou perdant. Pour résoudre ce problème, il est nécessaire d'avoir une fonction d'évaluation (une heuristique), capable d'estimer le plus précisément possible la qualité d'une configuration, et qui servira à évaluer les feuilles de l'arbre.

   Une fonction d'évaluation prend en entrée une configuration d'un jeu et donne en sortie une évaluation numérique (un entier ou un réel) représentant le score tel que :

   - Le signe indique qui a l'avantage (positif pour celui pour qui on calcule le meilleur coup, négatif pour l'autre)
   - La valeur absolue indique l'importance de cet avantage (pour avoir l'avantage, il faut maximiser son score et minimiser celui de l'adversaire).

   Plus l'heuristique sera précise et plus l'IA sera forte car évaluera mieux une position donnée.

   1. Proposer une heuristique dans le cas de l'Othello pour évaluer une position donnée. Justifiez votre choix dans le rapport.

   2. Implémenter une IA niveau intermédiaire en utilisant l'heuristique choisie à la question précédente. Pour cela, implémenter l'algorithme Min-Max en fixant la profondeur de recherche à 5 coups.

   3. Proposer une solution permettant de visualiser sur la console l'arbre de recherche (ou arbre des possibilités) ainsi que les coûts de chaque noeud.

2. **Niveau 2 : Alpha-Beta**

   Le problème principal de l'algorithme Min-Max est son temps de calcul qui peut être très long surtout si la profondeur de recherche est élevée.

   L'algorithme Alpha-Beta correspond à une amélioration de l'algorithme MinMax. Il permet de ne pas explorer l'arbre en intégralité en se basant sur la technique de *l'élagage* de l'arbre, c'est-à-dire l'élimination de certains nœuds de l'arbre. Les étapes principales de cet algorithme sont :
   - Définition de deux bornes : α et β
     - α est la borne inferieure de la valeur d'un nœud
     - β est la borne supérieure de la valeur d'un noeud
   - Initialisation: $\alpha=-\infty$, $\beta=+\infty$
   - Elagage des nœuds
     - Max si val courante > β
     - Min si val courante < α.
   - Les nœuds élagués sont ceux tels qu'évaluation(n) $\in [\alpha, \beta]$ et $\alpha \geq \beta$

Consignes importantes:

- **Ce projet doit être réalisé en binôme**, chacun des deux membres doit s'investir de la même façon, la note finale peut être différente entre les deux membres en fonction du degré d'investissement de chacun.

- **Langage de programmation :** le langage est imposé : java en POO obligatoire avec génération de la javadoc.

- **Versionning sous GIT** : Vous devez obligatoirement utiliser l'outil de versionning Git pour gérer votre projet (utiliser le serveur GitLab https://git-etudiants.lacl.fr). Lors de la soutenance, vous devrez montrer les différentes phases de développement de votre projet (avec des "révisions" hebdomadaires). Cela permettra de savoir "qui a fait quoi" dans le projet.
  Un document sera mis à votre disposition pour vous guider sur l'utilisation de GitLab.

- **Un rapport** d'environ 15 pages qui doit comprendre :

  - Une première page de couverture (titre, auteurs, année)
  - Une introduction avec Présentation rapide du projet
  - Les choix de programmation (structure de données)
  - Un diagramme de classe général de votre programme
  - Quelques fonctionnalités les plus pertinentes sous forme d'algorithmes
  - La répartition des tâches au sein de l'équipe
  - Les difficultés rencontrées
  - Une conclusion
  - Les citations de vos sources

- **Soutenances :** des soutenances seront organisées au retour des vacances de fin d'année, les dates exactes vous seront communiquées ultérieurement.

- **Interface utilisateur :** toutes les améliorations permettant d'améliorer l'interface utilisateur (utilisation de couleurs, déplacement simplifié au lieu d'une saisie …) sont les bienvenus et peuvent apporter des bonus.

Critères d'évaluation

- Votre travail sera jugé sur les critères suivants :

  - Qualité de conception
  - Qualité du code (organisation, javadoc, … etc)
  - L'intérêt, l'originalité, la jouabilité et toutes les caractéristiques importantes que vous devez mettre en avant lors de votre soutenance
  - La bonne répartition des tâches entre les membres de l'équipe
  - Qualité du Rapport

- Un code qui ne compile pas ou qui plante au démarrage ne vaut pas plus de 10/20. Tester donc votre programme avant de le déposer sur eprel-v2

- Le rendu se fera exclusivement sur eprel-v2 avant la date de soutenance (la deadline sera précisée sur eprel-v2 sur la page du cours « Projet programmation »). Tout retard sera pénalisé.

- Le barème indicatif sera réparti de la façon suivante :
  - Partie 1 : 8 points
  - Partie 2-A : 5 points
  - Partie 2-B : 5 points
  - Rapport : 2 points