M1 GIL- Mini-projet Théorie des jeux 2022-2023

Le jeu OThello (JAVA)

Module : Théorie des jeux
Responsable de cours : Carla Selmi
Rédigé par : • **KADA KELLOUCHA Samah** • **DJERMANI Hanane** • **Soualmi Madjda**
Date : 21/05/2023

Table des matières

Table des figures

1 Introduction

Jeu Othello, également connu sous le nom de Reversi, a été inventé au Japon en 1883. Il a été créé par un homme nommé Goro Hasegawa, qui l'a appelé Sekai no Hate Made (jusqu'aux confins du monde). Le jeu a été introduit aux États-Unis en 1975 et est rapidement devenu populaire dans le monde entier.

Le jeu se joue sur un plateau de 8x8 cases avec des pièces noires et blanches. L'objectif du jeu est d'avoir plus de pièces de votre couleur que votre adversaire à la fin de la partie. Les pièces sont capturées en les encadrant entre deux pièces de la couleur opposée, ce qui les retourne pour devenir de votre couleur.

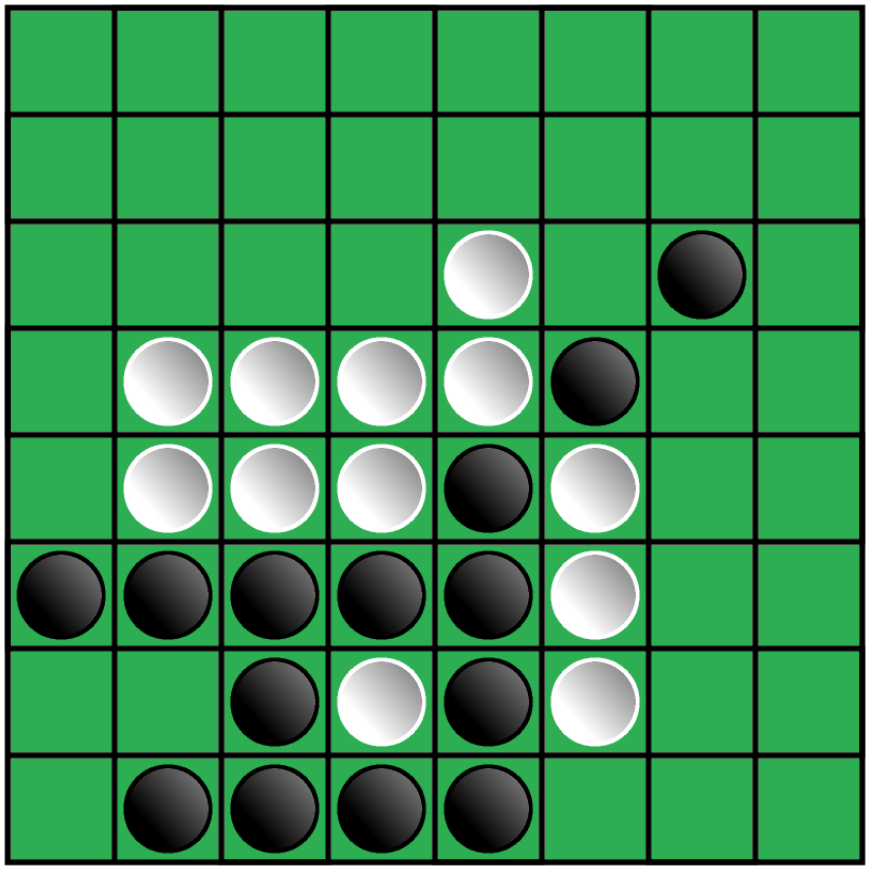

FIGURE 1 – Le jeu Othello

2 Travail demandé

Notre travail consiste à développer le jeu OTHELLO en utilisant différentes stratégies algorithmiques implémentées en Java.
Pour cela, nous allons programmer les algorithmes classiques de stratégies de jeu tels que **Min-Max**, **Alpha-Beta**, **A***, **NégaAlphaBeta** et **SSS***.
Pour une utilisation efficace de ces algorithmes, il est important de mener une analyse correcte du jeu.

2.1 Les règles de jeu Othello

Les règles du jeu Othello, également appelé Reversi, sont les suivantes :

- Le jeu est joué sur un plateau de 8x8 cases.
- Chaque joueur a un certain nombre de pions de sa couleur. Au début du jeu, deux pions noirs et deux pions blancs sont placés dans les cases centrales du plateau.
- Les joueurs jouent chacun leur tour. Le joueur avec les pions noirs commence la partie.
- Un coup consiste à placer un pion de sa propre couleur sur une case vide du plateau.
- Pour que le coup soit valide, le joueur doit placer son pion de manière à ce qu'il encercle un ou plusieurs pions adverses entre le pion placé et un autre pion de sa couleur déjà présent sur le plateau.
- Les pions encerclés sont alors retournés et deviennent de la couleur du joueur qui vient de jouer.
- Si un joueur ne peut pas jouer, il passe son tour. Si les deux joueurs passent leur tour consécutivement, la partie se termine.
- Le jeu se termine lorsque toutes les cases sont occupées ou qu'aucun des deux joueurs ne peut jouer.
- Le joueur qui a le plus de pions de sa couleur sur le plateau à la fin de la partie est déclaré vainqueur.

2.2 Les Algorithmes demandés

Min-Max [2] : Cet algorithme est une technique de recherche de l'arbre de jeu pour les jeux à deux joueurs. Il consiste à maximiser le gain possible du joueur en minimisant le gain possible de son adversaire. Pour chaque coup possible, il explore tous les coups suivants possibles et évalue le score obtenu pour le joueur en question. Ensuite, il choisit le coup qui maximise le score pour le joueur .

```
Fonction Minimax(noeud: Noeud, arborescence: Arborescence): Réel;
    Variable val: Réel;
    Variable k: Entier;

    Si noeud.Status = Feuille alors
        val ← noeud.h
    Sinon
        Si noeud.Etiquette = Max alors
            val ← -∞
            Pour k allant de 1 à bf faire
                val ← Maximum(val, Minimax(noeud.Fils[k], arborescence))
        Sinon
            val ← +∞
            Pour k allant de 1 à bf faire
                val ← Minimum(val, Minimax(noeud.Fils[k], arborescence))
        FinSi
    FinSi

    Retourner val
FinFonction
```

FIGURE 2 – Algorithme MINIMAX

Alpha-Beta [3] : C'est une amélioration de l'algorithme Min-Max qui permet de réduire considérablement le nombre de nœuds évalués. Au lieu d'explorer tous les nœuds, l'algorithme Alpha-Beta utilise une technique de coupure pour éviter d'explorer des branches qui ne contribuent pas au résultat final. L'algorithme utilise deux valeurs, alpha et beta, pour suivre les valeurs minimales et maximales possibles.

```
Fonction AlphaBeta(s: Noeud, Γ: Arborescence, α: Réel, β: Réel): Réel;
    Variable i: Entier;
    Variable val: Réel;

    Si s.Status = Feuille alors
        Retourner s.e
    Sinon
        Si s.Etiquette = Max alors
            i ← 1
            Tant que α < β et i ≤ k faire
                α ← Maximum(α, AlphaBeta(s.Fils[i], Γ, α, β))
                i ← i + 1
            FinTantQue
            Retourner α
        Sinon
            i ← 1
            Tant que α < β et i ≤ k faire
                β ← Minimum(β, AlphaBeta(s.Fils[i], Γ, α, β))
                i ← i + 1
            FinTantQue
            Retourner β
        FinSi
    FinSi
FinFonction
```

FIGURE 3 – Algorithme ALPHABETA

NégaAlphaBeta [4] : C'est une variation de l'algorithme Alpha-Beta qui est souvent utilisée pour les jeux à somme nulle comme l'othello. Il est similaire à l'algorithme Alpha-Beta, mais utilise une valeur de score négative pour le joueur adverse.

```
Fonction NegaAlphaBeta(s: Noeud, Γ: Arborescence, α: Réel, β: Réel): Réel;
    Variable i: Entier;
    Variable val: Réel;

    Si s.Status = Feuille alors
        Retourner s.g
    Sinon
        i ← 1
        val ← -∞
        Tant que α < β et i ≤ k faire
            val ← Maximum(val, -NegaAlphaBeta(s.Fils[i], Γ, -β, -α))
            α ← Maximum(α, val)
            i ← i + 1
        FinTantQue
        β ← Maximum(-α, β)
    FinSi

    Retourner val
FinFonction
```

FIGURE 4 – Algorithme NégaAlphaBeta

SSS* [5] : C'est une amélioration de l'algorithme A* qui utilise une liste ouverte de nœuds pour stocker les nœuds qui doivent encore être évalués. Contrairement à l'algorithme A*, qui utilise une file de priorité, SSS* utilise une liste liée. Cela permet de réduire le temps de calcul en évitant les opérations de tri nécessaires pour maintenir l'ordre de priorité dans une file de priorité.

```
Fonction SSS*(n: Arborescence, Γ): Réel;
    Variable i: Entier;
    Variable G: File;

    G ← ∅;
    Insérer((n, vivant, +∞), G);

    Tant que Premier(G) n'est pas de la forme (n, résolu, v) faire
        (n, t, e) ← Extraire-Premier(G);

        Si t = vivant alors
            Si n est terminal alors
                Insérer((n, résolu, Minimum(e, h(n))), G)
            Sinon
                Si n est de type Max alors
                    i ← 1;
                    Pour i de 1 à k faire
                        Insérer((fi(n), vivant, e), G)
                        i ← i + 1;
                    FinPour
                Sinon {n est de type Min}
                    Insérer(fgne(n), vivant, e), G
                FinSi
```

```
        Sinon {t est résolu}
            Si n est de type Min alors
                Insérer((p(n), résolu, e), G);
                Supprimer de G tous les états dont le noeud est un successeu
            Sinon {n est de type Max}
                Si n a un frère droit alors
                    Insérer((f_rdne(n), vivant, e), G)
                Sinon
                    Insérer((p(n), résolu, e), G)
                FinSi
        FinTantQue

        (n, t, e) ← Extraire-Premier(G);
    FinTantQue

    Retourner(e)
FinFonction
```

FIGURE 5 – Algorithme SSS*

A* : C'est un algorithme de recherche de chemin qui peut être utilisé pour trouver le chemin optimal entre deux points dans un graphe. Il utilise une fonction heuristique pour estimer le coût du chemin restant à partir du nœud actuel. L'algorithme explore les nœuds en fonction de leur coût total, qui est la somme du coût actuel et du coût heuristique estimé.

```
Procédure A*(G: Graphe, s, t: Sommet; var Δ, n, O, f);
    Début
    Init*(G, s, Δ, n, O, f);
    x ← s;

    Tant que (x ≠ t) et (O ≠ ∅) faire
        Début
        Examiner*(x, Δ, n, O, f);

        Si (O ≠ ∅) alors
            Choisir dans O un sommet z d'approximation f(z) minimale;
            x ← z;
        FinSi
    FinTantQue

    FinProcédure
```

FIGURE 6 – Algorithme A*

3 Document technique

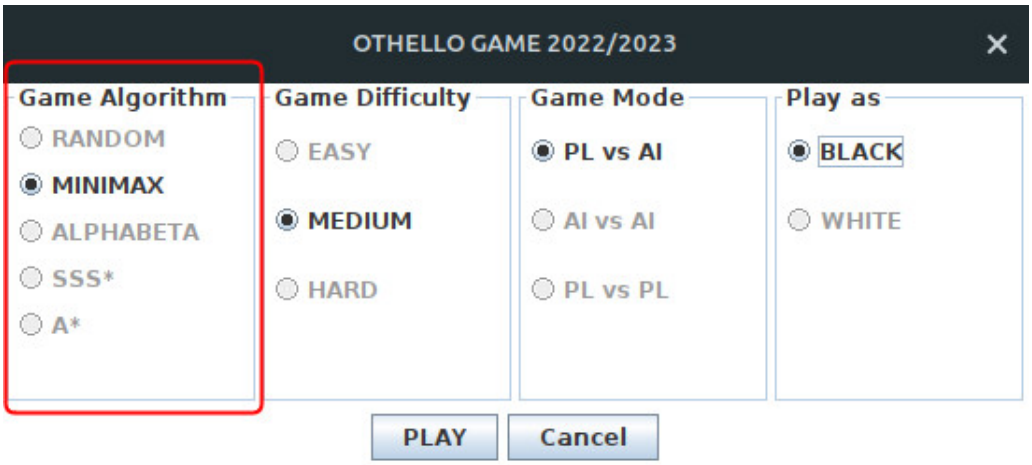

FIGURE 7 – Choix d'algorithme

FIGURE 8 – Choix de niveau

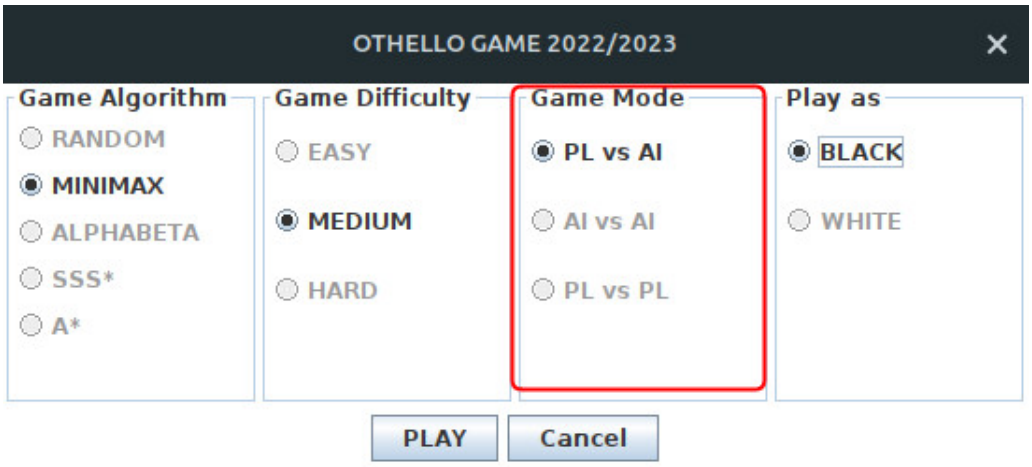

FIGURE 9 – Choix de mode

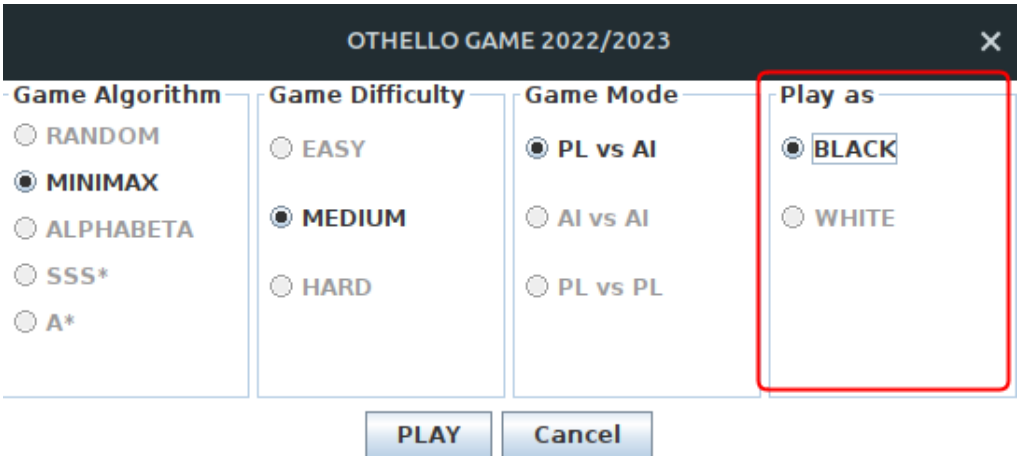

FIGURE 10 – Choix de type

4 Conclusion

En conclusion, notre projet a consisté à réaliser le jeu Othello en Java en utilisant différents algorithmes de stratégies de jeu tels que MinMax, AlphaBeta, NegAlphaBeta, SSS* et A* malheureusement on à réussi à implémenté que les 4 premiers . Nous avons implémenté les règles du jeu et avons offert la possibilité aux joueurs de choisir leur niveau de difficulté ainsi que leur mode de jeu **(joueur contre joueur, joueur contre ordinateur, ordinateur contre ordinateur)**. Pour être efficaces, nous avons basé l'utilisation de ces algorithmes sur une analyse correcte du jeu. Ce projet nous a permis de renforcer nos compétences en programmation orientée objet ainsi que notre compréhension de l'utilisation des algorithmes pour la résolution de problèmes.