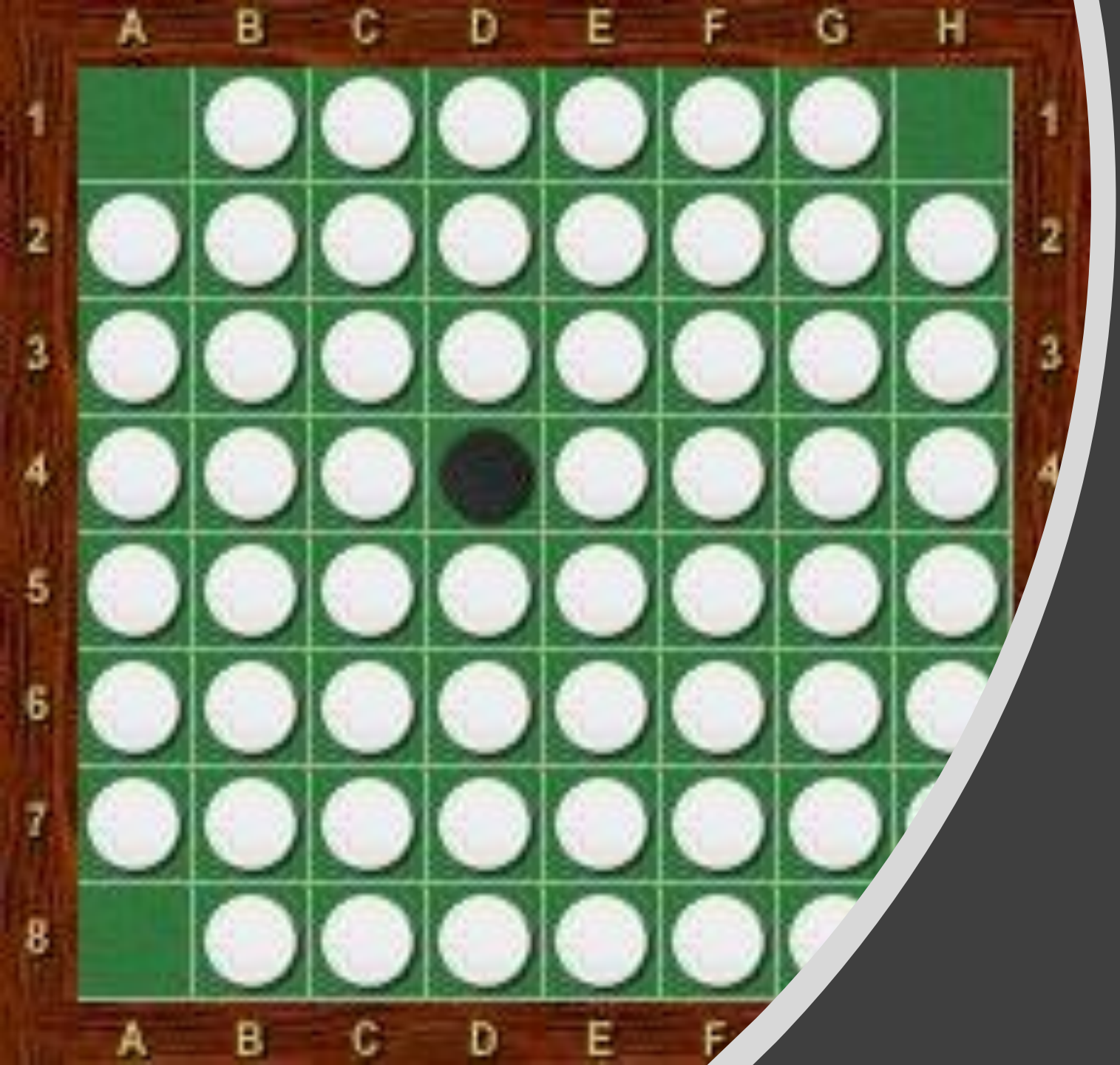


Othello

AL RAYYAN Lawrence
AIDOUNE Leaticia
2eme année à l’upec / 2022

PRESENTATION DU JEU OTHELLO

- L'Othello est un jeu de plateau de stratégie pour deux joueurs. Le plateau de jeu est constitué de 64 cases de couleur noire et blanche disposées en un damier. Chaque joueur a une couleur de pion, noir ou blanc. Le but du jeu est d'avoir le plus grand nombre de pions de sa couleur sur le plateau à la fin de la partie.
- Le jeu commence avec quatre pions placés au centre du plateau, deux pions noirs et deux pions blancs. Les pions sont placés de manière à ce que les coins opposés du plateau soient de couleur noire.
- Le joueur qui a les pions noirs joue en premier. Les joueurs jouent à tour de rôle, en plaçant un pion de leur couleur sur le plateau, en prenant soin de retourner un ou plusieurs pions de l'adversaire au passage. Pour retourner les pions de l'adversaire, il faut que le pion placé soit adjacent à un pion de l'adversaire et qu'il y ait une rangée de pions de l'adversaire de la même couleur à l'autre extrémité de la rangée.
- La partie se termine lorsqu'il n'y a plus de cases disponibles sur le plateau ou lorsqu'aucun joueur ne peut plus jouer de coup valide. Le gagnant est celui qui a le plus grand nombre de pions de sa couleur sur le plateau à la fin de la partie.

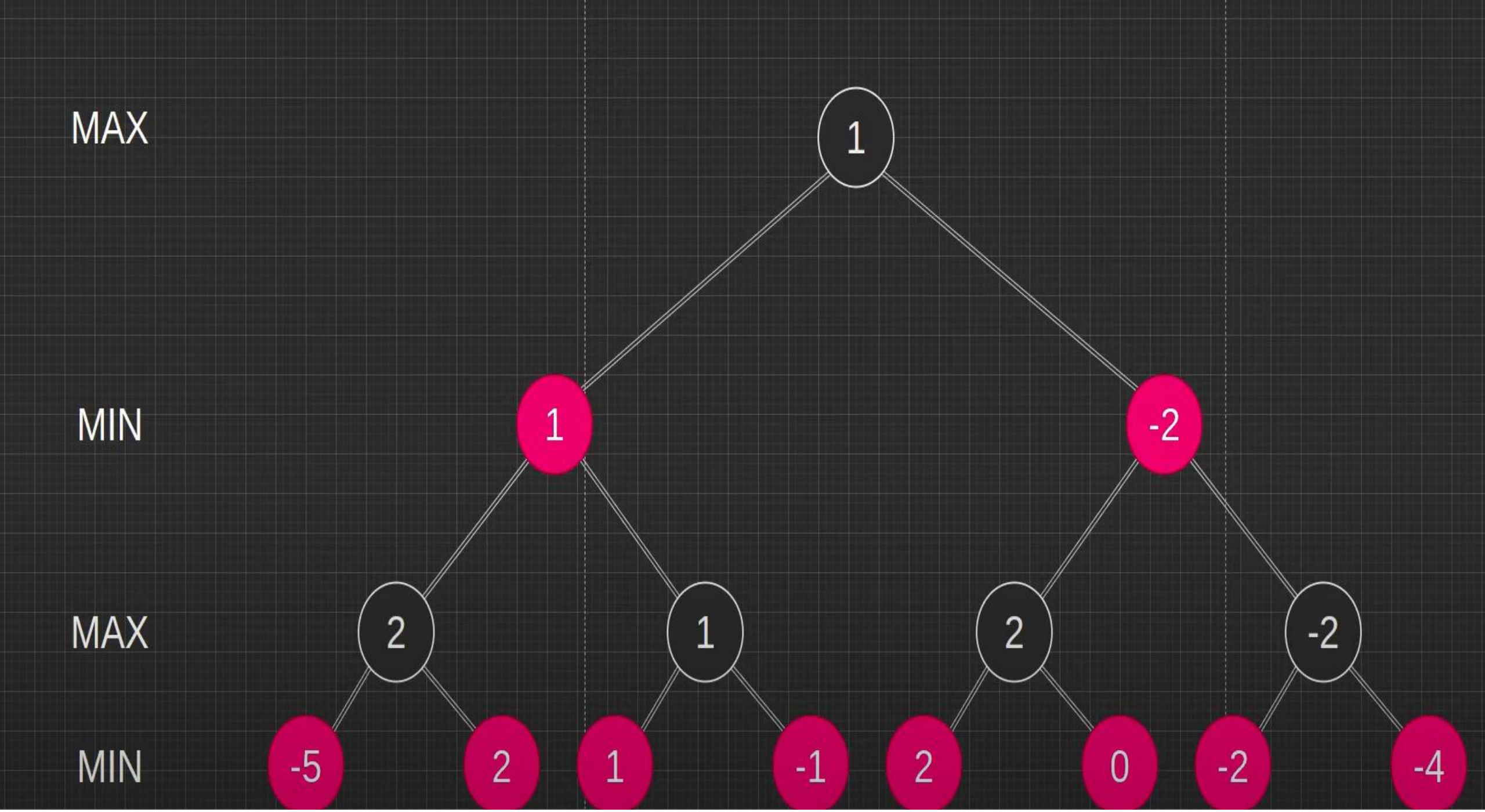
PRESENTATION DU PROGRAMME

- Le programme utilise Java / Swing pour l’interface graphique
- Il permet de jouer à deux ou contre un Intelligence Artificiel (MinMax) ou Intelligence Artificiel naif
- L'algorithme MinMax est un algorithme de recherche utilisé dans les jeux à deux joueurs pour trouver le coup optimal à jouer. Il est basé sur l'idée de minimiser les pertes et maximiser les gains à chaque tour.

Voici comment il fonctionne :

1. L'algorithme commence par l'état courant du jeu, génère tous les coups possibles à partir de cet état et évalue leur valeur.
2. Pour chaque coup possible, l'algorithme simule le coup et calcule la valeur de la meilleure réponse possible de l'adversaire.
3. L'algorithme choisit alors le coup qui minimise la perte ou maximise le gain en fonction de si c'est à lui ou à l'adversaire de jouer.
4. Le processus est répété jusqu'à ce qu'un état terminal soit atteint, c'est-à-dire que la partie soit terminée.

L'algorithme MinMax est généralement utilisé avec une fonction d'évaluation pour évaluer la valeur des états du jeu. Cette fonction prend en compte les forces et les faiblesses de chaque joueur pour déterminer lequel est le plus susceptible de gagner.


MAX
MIN
MAX
MIN
1
1
-2
2
1
2
-2
-5
2
1
-1
2
0
-2
-4

Les choix de programmation (structure de données)

- Un tableau d’entière pour représenter le damier
- Une ArrayList<int[][]> une liste qui contient des tableaux à deux dimensions d'entiers. Cela peut être utile pour stocker une série de grilles de jeu, annuler le mouvement et pour sauvegarder le damier

Voici comment on peut déclarer une ArrayList<int[][]> et y ajouter des éléments :

```
ArrayList<int[][]> grids = new ArrayList<int[][]>();

// Ajouter un tableau à deux dimensions à la liste
int[][] grid1 = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
grids.add(grid1);

// Ajouter un autre tableau à deux dimensions à la liste
int[][] grid2 = {{10, 11, 12}, {13, 14, 15}, {16, 17, 18}};
grids.add(grid2);
```

Le tableau **offsets** déclaré est un tableau à deux dimensions qui contient des couples de coordonnées (x, y) utilisés pour déplacer un point dans une grille en utilisant les déplacements possibles suivants :

- (0, 1) : déplacement vers la droite
- (1, 0) : déplacement vers le bas
- (0, -1) : déplacement vers la gauche
- (-1, 0) : déplacement vers le haut
- (1, 1) : déplacement en diagonal vers la droite et le bas
- (1, -1) : déplacement en diagonal vers la droite et le haut
- (-1, -1) : déplacement en diagonal vers la gauche et le haut
- (-1, 1) : déplacement en diagonal vers la gauche et le bas

Ces déplacements peuvent être utilisés pour parcourir une grille de manière adjacente, en partant d'un point donné. Par exemple, si on veut parcourir tous les points adjacents à (x, y) dans une grille, on peut utiliser une boucle « for » pour parcourir le tableau « offsets » et ajouter chaque élément au point de départ :

```
int x = 3;
int y = 4;

for (int[] offset : offsets) {
  int newX = x + offset[0];
  int newY = y + offset[1];
  // faire quelque chose avec les nouvelles coordonnées (newX, newY)
}
```

Cela permet de parcourir tous les points adjacents à (x, y) dans la grille, y compris les points en diagonal.

Méthode principaux :

- « LeVoisineContient » :

Prend en entrée un entier cible « target », ainsi que des coordonnées de ligne et de colonne (row et col) et vérifie si un des voisins de la cellule située à ces coordonnées contient la valeur cible.

Pour ce faire, la méthode parcourt le tableau offsets déclaré précédemment et ajoute chaque élément du tableau aux coordonnées de départ (row et col) pour obtenir les coordonnées des voisins.

Si l'une des cellules voisines contient la valeur cible, la méthode retourne true.

Si aucune cellule voisine ne contient la valeur cible, la méthode retourne false.

```
int target = 5;
int row = 3;
int col = 4;

if (LeVoisineContient(target, row, col)) {
  // faire quelque chose si l'un des voisins contient la valeur cible
} else {
  // faire quelque chose si aucun des voisins ne contient la valeur cible
}
```

- «getLegalMove » :

retourne une liste de coups légaux qui peuvent être joués. Pour trouver ces coups légaux, la méthode parcourt toutes les cellules de la grille et vérifie si elles remplissent les critères suivants :

- La cellule doit être vide (c'est-à-dire avoir la valeur 0).
- Au moins un des voisins de la cellule doit être occupé par une pièce de l'adversaire (c'est-à-dire avoir une valeur différente de turn).

Si une cellule remplit ces critères, la méthode parcourt les lignes, les colonnes et les diagonales qui passent par cette cellule pour vérifier si elle peut être jouée légalement.

Pour être légale, la cellule doit être entourée par au moins une pièce de la même couleur que turn sur la ligne, la colonne ou la diagonale qui passe par cette cellule. Si c'est le cas, la cellule est ajoutée à la liste de coups légaux et la boucle est interrompue afin de passer à la cellule suivante.

Une fois que toutes les cellules de la grille ont été vérifiées, la méthode retourne la liste de coups légaux.

```
ArrayList<Integer[]> legalMoves = getLegalMoves();

for (Integer[] move : legalMoves) {
  int row = move[0];
  int col = move[1];
  // faire quelque chose avec les coups légaux (row, col)
}
```

- « estLegal » :

vérifie si un coup spécifié par ses coordonnées de ligne et de colonne (row et col) est légal dans un jeu de "Reversi" (également connu sous le nom d'Othello).

Pour ce faire, la méthode utilise la méthode getLegalMoves décrite précédemment pour obtenir la liste de tous les coups légaux possibles et vérifie si le coup spécifié s'y trouve.

Si le coup est dans la liste, la méthode retourne true, sinon elle retourne false.

```
int row = 3;
int col = 4;

if (estLegal(row, col)) {
  // faire quelque chose si le coup (row, col) est légal
} else {
  // faire quelque chose si le coup (row, col) n'est pas légal
}
```

- « playMove » :

Elle prend en entrée les coordonnées de ligne et de colonne (row et col) du coup à jouer.

Avant de jouer le coup, la méthode vérifie si celui-ci est légal en appelant la méthode estLegal.

Si le coup est légal, la méthode joue le coup en plaçant une pièce de la couleur du joueur actuel (déterminée par la variable turn) dans la cellule spécifiée.

Ensuite, la méthode parcourt les lignes, les colonnes et les diagonales qui passent par la cellule jouée et vérifie si elles contiennent des pièces de l'adversaire qui peuvent être retournées.

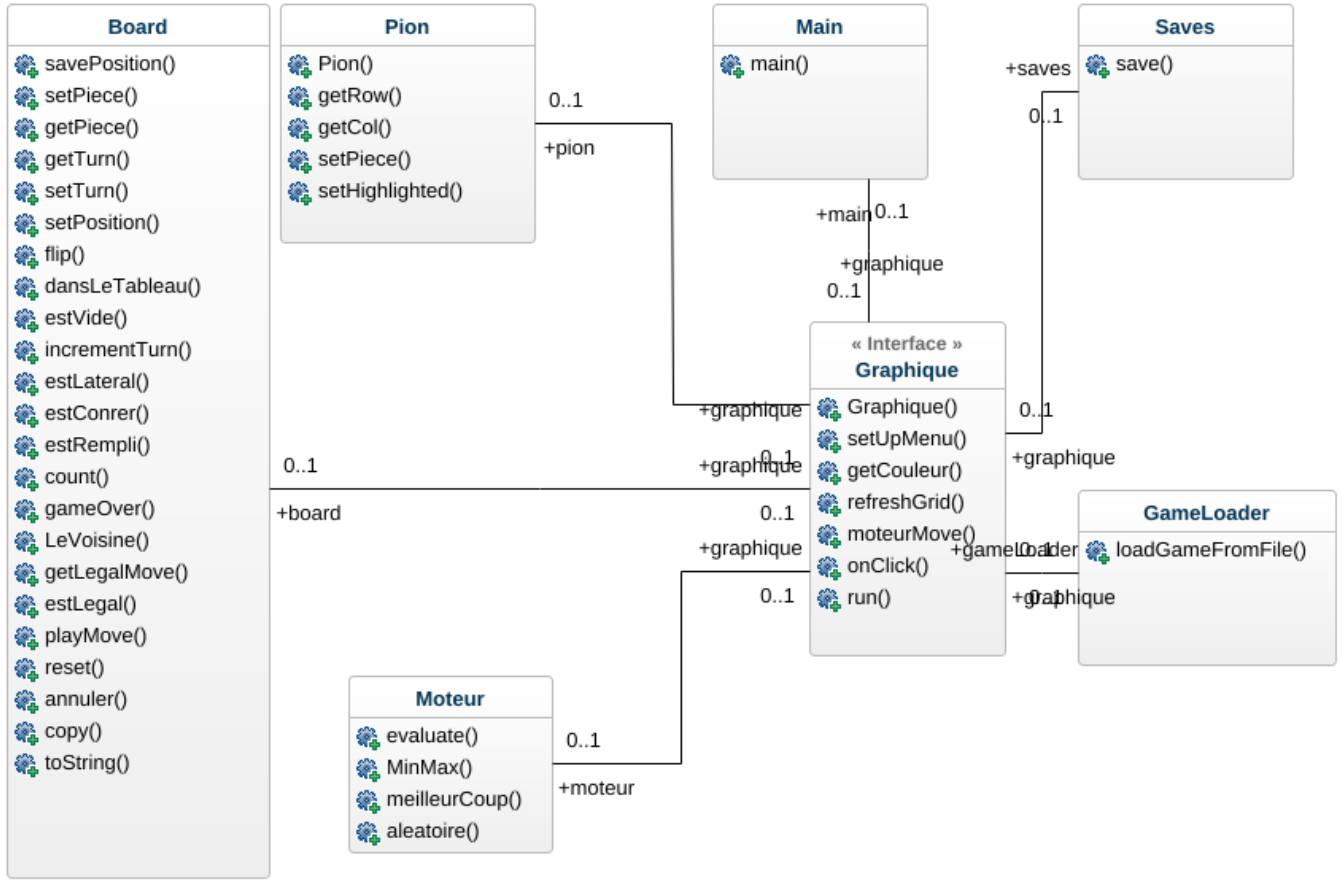
Si c'est le cas, la méthode retourne toutes ces pièces en utilisant la méthode flip.

Enfin, la méthode enregistre la position actuelle de la grille dans un historique de positions et inverse la valeur de turn pour indiquer que c'est maintenant au tour de l'autre joueur de jouer.

```
int row = 3;
int col = 4;

playMove(row, col);
```

Class-Diagrame



La répartition des tâches

Il est important de bien répartir les tâches au sein de l'équipe pour assurer l'efficacité et la productivité de l'équipe, et il est important de donner des tâches qui correspondent aux compétences et aux connaissances de chaque membre de l'équipe, afin de maximiser leur contribution au projet.

Il est également important de veiller à ce que les tâches soient réparties de manière équitable entre les membres de l'équipe, afin d'éviter que certains membres soient surchargés de travail tandis que d'autres seront inactifs.

De ce fait, nous avons décidé de tout réaliser ensemble. Nous avons fait le choix de nous rejoindre dans un espace commun afin d’avancer le travail.

Les difficultés rencontrées

1. La sauvegarde et le chargement d'une partie : c’était difficile de mettre en place une fonctionnalité de sauvegarde et de chargement de partie de manière efficace et fiable en tenant compte de tous les aspects du jeu qui doivent être sauvegardés (état de la grille, score, tour de jeu, etc.) et trouver un moyen de les stocker de manière à pouvoir les charger à tout moment.
2. Vérifier les coups possibles : c’est important de vérifier quels coups sont légalement possibles pour chaque joueur à chaque tour, afin d'éviter les erreurs de jeu.
3. MinMax : L'algorithme de min-max était complexe à mettre en place et à optimiser
4. L'interface graphique : la création d'une interface graphique était un défi, surtout qu’on n’a pas expérience en design d'interface utilisateur.

En résumé, la création du était un défi pour l’équipe, en raison de la complexité de certaines tâches comme la sauvegarde et le chargement de partie, la vérification des coups possibles, la création de l'intelligence artificielle et l'interface graphique, et la soumission du travail sur GitLab. Toutefois, en travaillant ensemble et en s'entraidant, l'équipe était en mesure de surmonter ces difficultés et de créer un jeu de qualité.

conclusion :

- Nous avons travaillé en binôme sur le développement d'un jeu de l'Othello en Java. Nous avons commencé par implémenter les règles du jeu, en permettant aux joueurs de poser leurs pions et de retourner les pions adverses selon les règles du jeu. Nous avons également implémenté une intelligence artificielle pour permettre à l'ordinateur de jouer contre l'utilisateur.
- Ensuite, nous avons ajouté la possibilité de sauvegarder et charger une partie en cours, en utilisant la bibliothèque ObjectOutputStream et ObjectInputStream de Java.
- Nous avons également implémenté une méthode de conclusion de jeu permettant de déterminer le gagnant de la partie en comptant le nombre de pions de chaque joueur sur la grille de jeu.
- Enfin, nous avons ajouté une interface graphique à notre jeu en utilisant la bibliothèque Swing de Java, pour rendre notre jeu plus agréable à utiliser.
- En résumé, nous avons développé un jeu de l'Othello complet en Java, proposant une intelligence artificielle, la possibilité de sauvegarder et charger une partie, et une interface graphique agréable. Nous sommes satisfaits du résultat final et nous espérons que notre jeu sera apprécié des joueurs.

Sources :

- Pour la bibliothèque Graphique :

https://www.youtube.com/@DominiqueLiard06

https://www.youtube.com/watch?v=KFPNqtjOHBI

https://www.youtube.com/watch?v=Yj8dLiMB4VM

- Pour le MinMax :

https://www.youtube.com/watch?v=YWolTTaFauk&t=1827s

https://www.youtube.com/watch?v=fXQfJro7yJ8

Exemple des jeux avec MinMax : https://www.youtube.com/watch?v=Q4RvgL3ic5Q&t=1211s

- Pour le sauvegarde :

https://www.youtube.com/watch?v=7hZVRDxpbCE&t=225s

https://www.youtube.com/watch?v=DU2VpMIzdFY

- Essayer le jeu afin de mieux comprendre : https://www.eothello.com/