

Projet jeu Othello

Dombry Baptiste
Dourlen Maxime
Ferkane Amazigh
Groupe TP3

Introduction

Au cours de notre dernier semestre de licence, nous devions réaliser un projet à plusieurs pendant tout le semestre sur un sujet au choix donné par les professeurs. Le projet Othello de M.Caron nous a tout de suite plu, car nous aimons bien les jeux de société ainsi que l'aspect jeux vidéo à laquelle on pensait du développement informatique.

L'Othello est un jeu de société combinatoire abstrait opposant deux joueurs, il se joue sur un tablier unicolore de 64 cases, 8 sur 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 des règles précises. 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.

Ce projet aura pour mission de confirmer nos apprentissages de la licence informatique, car nous allons utiliser beaucoup de notions que nous avons vues de la première à la troisième année.

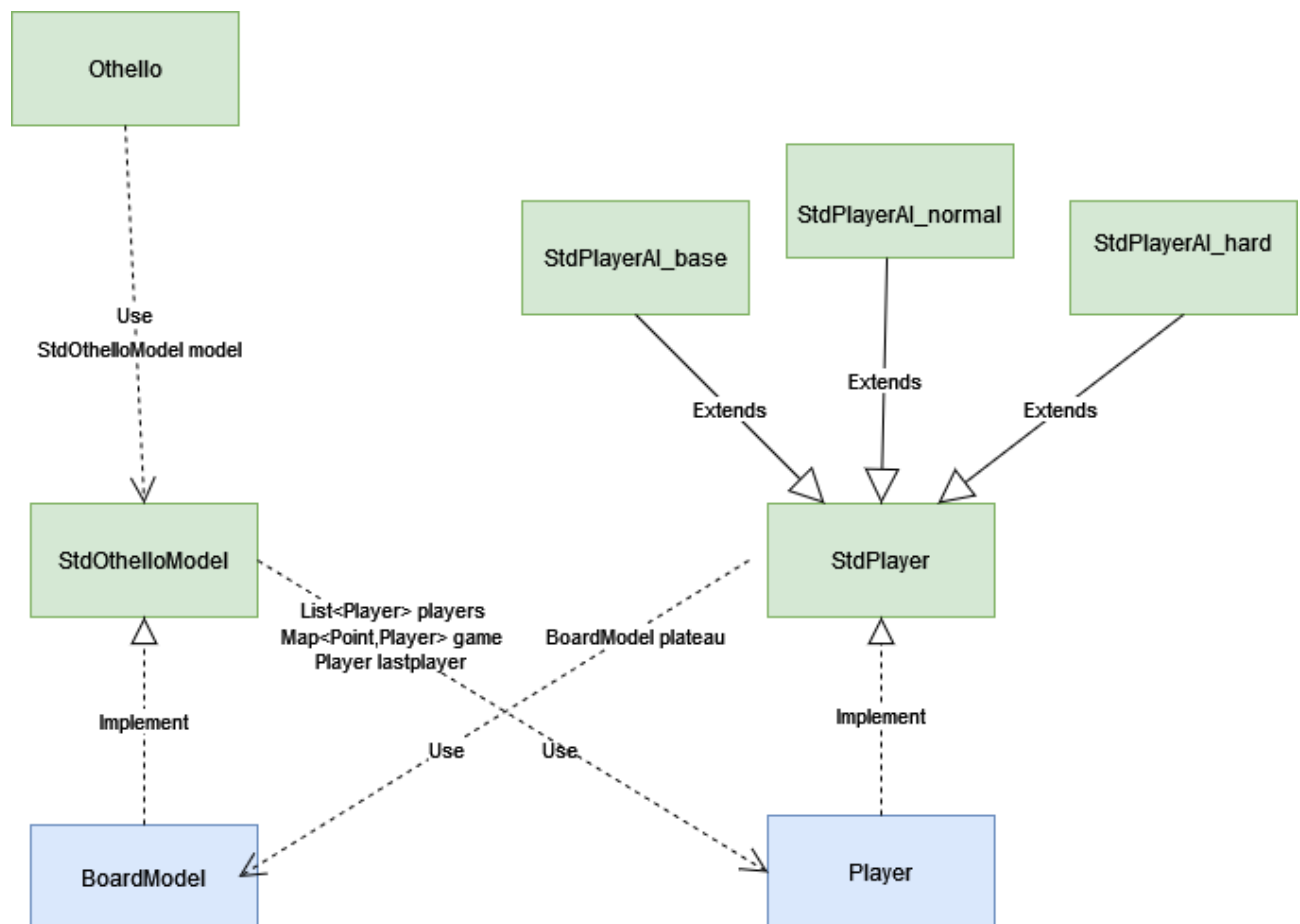
Nous avons choisi de développer cette application dans le langage Java dans le cadre de la suite de notre cursus en méthodologie de la programmation orientée objet ainsi que le cursus d'interface homme-machine. Nous allons présenter en premier le modèle du plateau et des joueurs, ensuite nous allons expliquer les choix graphiques et pour finir nous allons mettre en avant les algorithmes utilisés.

Sommaire

Introduction	2
Modèle	4
Interface	5
La barre d'outils	5
Le plateau de jeu	5
Les infos de jeu, Bouton de lancement/fin	6
Algorithmes	7
Liste des coups possibles	7
IA Facile	8
IA Moyen	8
IA Difficile	9
Difficultés rencontrées	11
Conclusion	12

Modèle

Pour créer le modèle, l'architecture de notre application, nous avons choisi d'utiliser les mêmes méthodes que celles vues dans les enseignements « méthodologie de la programmation orientée objet 2 » ainsi que celles vues en « Interface Homme-Machine ». Pour cela, nous avons choisi de créer des interfaces de classe qui représentent les objets dont nous aurons besoin. Les interfaces « BoardModel » et « Player » représentent des objets génériques avec les fonctions minimales à intégrer pour le bon fonctionnement de l'application. Comme nous pouvons le voir sur l'image ci-dessous, toutes les classes découlent des interfaces citées précédemment.



Les interfaces « BoardModel » et « Player » ont été développées de manière à ce qu'elles soient utilisables pour différents jeux. L'othellier faisant 8 cases de large et de hauteur, il serait possible par exemple de développer

grâce à l'interface « BoardModel », une classe « StdChestModel » qui représenterait un jeu d'échecs.

Interface

Pour l'interface, nous avons décidé d'utiliser Java Swing en continuité à notre cours d'Interface Homme Machine du sixième semestre de Licence. L'application se présente comme ceci :

On peut la séparer en 3 parties :

La barre d'outils

La barre d'outils est composée de 3 éléments, « Nouvelle partie », « Option de jeu » et « Info ». L'élément « Nouvelle partie » va ouvrir un menu déroulant et nous pourrions choisir les 4 modes de jeux disponibles dans l'application, le mode 2 joueurs, ainsi que le mode 1 joueur contre l'IA, qui peut être facile, normale ou difficile. L'élément « Option de jeu » ouvre un menu déroulant donnant les options de jeu. Ici on a uniquement l'option pour connaître les cases où l'on peut placer le prochain pion, c'est un axe de travail futur pour ajouter des fonctionnalités à l'application. Ensuite, le dernier élément est « Info » qui permet lorsque l'on clique dessus, d'ouvrir la notice de l'application ainsi que les règles du jeu.

Pour la barre d'outils, les composants Swing utilisés sont les JMenuBar, les JMenu et les JMenuItem. La classe JMenuBar est utilisée pour afficher la barre de menu sur la fenêtre. Il peut y avoir plusieurs menus. L'objet de la classe JMenu permet de créer un menu déroulant qui est affiché à partir de la barre de menus. Il hérite de la classe JMenuItem. L'objet de la classe JMenuItem ajoute un simple élément au menu. Les éléments utilisés dans un menu doivent appartenir au JMenuItem ou à l'une de ses sous-classes.

Le plateau de jeu

Le plateau de jeu est la partie centrale de l'application et nécessairement la plus importante, car c'est sur celle-là que l'on joue. Le plateau est représenté par des boutons de couleur verte (la même couleur que l'othellier). À ces boutons on associe des `ActionListeners`, qui permettent de faire certaines actions dès que l'on clique dessus, ici les boutons ou les coups ne sont pas possible sont désactivés. À chaque fois que c'est au tour du joueur, quand il appuiera sur le bouton, cela posera son pion tout en retournant les pions de la couleur différente qu'il a encadré, par un système de rafraîchissement de cases.

Java `ActionListener` est notifié à chaque fois que vous cliquez sur le bouton. Il est notifié contre `ActionEvent`. L'interface `ActionListener` se trouve dans le package `java.awt.event`. Il n'a qu'une seule méthode `actionPerformed()`. La méthode `actionPerformed()` est invoquée automatiquement chaque fois que vous cliquez sur le bouton.

Le tableau de boutons implémente aussi la classe `StdBoardModel`.

Les infos de jeu, Bouton de lancement/fin

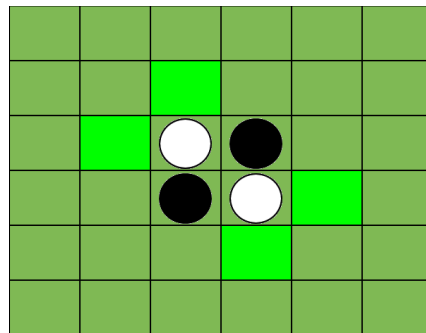
Pour finir en dessous du plateau, se trouve le tour de la personne qui doit jouer, suivi de boutons pour lancer ou recommencer une partie. Les boutons sont sécurisés, on ne peut pas lancer de partie quand elle est déjà lancée ou quand on n'a pas choisi de mode de jeu.

Les boutons sont créés à l'aide des composants Swing `JButton`, la classe `JButton` est utilisée pour créer un bouton étiqueté ayant une implémentation indépendante de la plateforme. L'application entraîne une action lorsque le bouton est cliqué. Il peut être configuré pour avoir différentes actions, en utilisant « `Event Listener` ». `JButton` hérite de la classe `AbstractButton`.

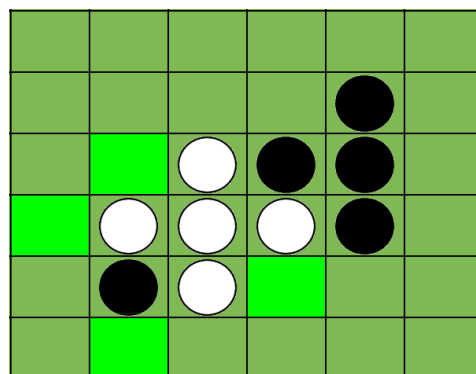
Algorithmes

Liste des coups possibles

Pour calculer la liste des coups possibles pour un certain joueur, nous récupérons d'abord la liste des coups qu'il a joués. Ensuite pour chaque coup joué par le joueur, on regarde s'il existe des pions de l'adversaire collés. Si un de ces pions existe alors on vérifie dans la continuité de l'axe si sans rencontrer un pion allié, il y a un emplacement vide. Dans ce cas, tous les pions passés par l'algorithme sont enregistrés dans une Map correspondant à cet emplacement vide. Cela est effectué pour tous les axes, diagonal, horizontal et vertical. Comme nous pouvons le voir sur l'image ci-dessous, les cases en vert sont les coups possibles.



Ces quatre cases correspondent au deux pions noirs. Le pion noir en haut à droite a deux coups jouables, un à sa gauche et un autre en bas. Comme on peut le voir, ce dernier est collé par deux pions adverses qui sur le même axe sont suivis par un emplacement vide qui devient alors un coup jouable.



Sur l'image ci-dessus, nous pouvons voir qu'un coup jouable peut être issu de plusieurs pions, comme le coup jouable le plus à droite. Ce dernier, s'il est joué, retournera les pions au-dessus et à gauche jusqu'au pion noir.

IA Facile

Pour le niveau facile de l'IA, il joue totalement aléatoirement sur la liste des coups possibles. Dans le code, on prend une liste des coups possibles, et on génère un nombre aléatoire compris entre 0 et la taille de la liste, puis on choisit l'élément dans la liste correspondant au numéro aléatoire. Son algorithme est le suivant.

Algorithm 1 IA Facile






```
L ← ListeCoupsPossible()
if L = 0 then
    return Aucun coups possibles
else
    return Random(L)
end if
```

IA Moyen

Pour le niveau moyen de l'IA, on a cherché les conseils proposés par les sites orientant les joueurs pour améliorer leur niveau de jeu du niveau débutant au niveau intermédiaire, on a trouvé sur le site www.ffothello.org cette liste :

Principes généraux de jeu dans les ouvertures

Voici les « grandes règles » (triées plus ou moins par ordre de priorité décroissante) que la théorie classique conseille pour jouer les ouvertures :

-  Essayez d'avoir *moins* de pions que votre adversaire.
-  Essayez de retourner le *centre* de la position (les 4 cases initiales dans les premiers coups).
-  Évitez de retourner trop de pions en *frontière* (c'est-à-dire les pions à l'extérieur de la position) et donc de vous construire des *murs*.
-  Essayez de regrouper vos pions en une seule masse plutôt que d'avoir des pions isolés et dispersés.
-  Évitez de prendre des bords trop tôt (avant le milieu de partie).

et donc nous avons en conséquence , développé la classe de telle manière à implanter ce système comme tel .

IA Difficile

Pour le niveau difficile de l' IA, on a décidé d'utiliser la force d'une position par la somme des valeurs des cases occupées par cette couleur.

Pour le calcul du critère de force, on doit attribuer à chaque case une valeur tactique qui représente l'intérêt qu'on a à l'occuper. Une valuation possible des cases est donnée (figure ci-dessous).

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

On peut justifier ce tableau par ces quelques remarques :

- Un pion placé dans un coin est imprenable et constitue donc une solide base de départ pour la conquête des bords.
- Les cases bordant le coin sont à éviter, car elles donnent à l'adversaire la possibilité de prendre le coin.
- Les cases centrales augmentent les possibilités de jeu.
- Les cases du bord sont également des points d'appui solides. On doit toutefois corriger cette évaluation positionnelle.
- Lorsqu'on occupe déjà un coin, la possession des trois cases voisines devient intéressante.
- Les cases du bord qui sont reliées au coin par une chaîne continue de pions de la même couleur deviennent plus intéressantes, car elles sont désormais imprenables.

L'algorithme prend à chaque tour de l'IA la liste des coups possibles et joue sur la case avec le poids le plus haut

Dans le code, on a dû créer le tableau ci-dessus, et faire correspondre la liste des coups possibles avec les poids des coups afin de choisir celui qui a le plus de poids. Celui qui aura le plus grand poids sera joué. Son algorithme est le suivant.

Algorithm 2 IA Difficile

```
L ← ListeCoupsPossible()
Max = Point()
lastmax = -250
TAB = InitMatrice()
for all P dans L do
    if TAB[P.X][P.Y] ≥ lastmax then
        lastmax = TAB[P.X][P.Y]
        max = P
    end if
end for
return max
```

Difficultés rencontrées

Les difficultés que l'on a rencontrées sont : l'organisation et la répartition du travail, la création d'une interface ergonomique.

Plus techniquement, la gestion d'un Thread d'attente. En effet, cette tâche est liée uniquement au coup joué par l'ordinateur. Ce coup serait effectivement trop rapide pour comprendre où il a été placé parce qu'il serait joué instantanément après le coup de l'autre joueur. Nous avons donc dû ajouter un délai entre le moment où le joueur joue et le moment où l'ordinateur joue. Ce délai bloquait l'entièreté de l'application. Étant sur le même thread, nous avons donc choisi de faire jouer l'IA dans un thread temporaire qui ne fait que choisir le coup et demander au Thread principal de jouer le coup voulu.

La recherche des algorithmes à utiliser pour trouver des algorithmes compatibles avec notre implantation actuelle du jeu, mais aussi les algorithmes les plus efficaces. La fusion des travaux de chacun était aussi une difficulté importante car quand nous travaillions en même temps il était difficile de regrouper les ajouts et les suppressions sans en oublier.

Il y avait des changements imprévus par exemple pour la classe IA difficile. Étant partis à la base sur un algorithme de calcul de probabilités pour anticiper le prochain coup de l'adversaire, nous n'avions pas pu gérer correctement le calcul à cause d'un problème d'héritage, donc nous avons trouvé un autre moyen de le faire.

Conclusion

Ce projet, basé sur le jeu d'Othello, est très intéressant car il permet d'appliquer à la fois nos connaissances en Java et en algorithmique. De plus, le jeu d'Othello est un jeu que nous connaissions tous les trois avant ce projet, ce qui nous a facilité l'analyse et la compréhension du sujet.

Nous avons appris les axes principaux de l'intelligence artificielle. En effet, notre exemple d'application n'est en fait qu'un exemple très simple dans lequel ce type de technologie peut être utilisé. Pour la suite du travail d'IA on pourrait implémenter l'algorithme de Mini-max qui est réputé comme une IA très forte.

Ainsi, le jeu fournit l'expérience attendue d'un jeu de réflexion. En effet, outre le programme adversaire, le jeu a subi au cours de la construction du projet, des modifications ergonomiques pour améliorer l'expérience de jeu. Bien que cela n'ait pas bénéficié de notre pleine attention, nous voulions fournir une interface de jeu acceptable et ludique.

Nous avons pris beaucoup de plaisir à coder ce jeu, et en prenons également à jouer à cette version de l'Othello