Rapport Projet Reversi

CHEVALIER Simon
LACROIX Florent

Introduction :

Ce projet a pour but de proposer à l'utilisateur de jouer au jeu du **Reversi** (plus précisément à l'Othello, étant donné que, par choix, nous avons préféré programmer une version numérique de l'Othello, car nous en préférions les règles (quelques variantes par rapport au Reversi classique)). Le principe est de retourner un maximum de pions de sa couleur. Quand une couleur encadre un ou plusieurs pions adverses par deux pions de sa couleur, alors tous les pions compris entre se « retournent». Le jeu se termine quand aucun des deux joueurs ne peut plus jouer. Il jeu se joue sur un plateau 8*8 cases.

Le jeu dispose de deux modes : *Player1 VS Player2* ou *Player VS Computer*. L'intelligence artificielle a été programmée de la sorte : quand l'ordinateur peut jouer, il connaît les cases sur lesquelles il est autorisé à mettre un pion. Sachant que les stratégies gagnantes ne sont pas celles qui retournent un maximum de pions à chaque tour, nous avons choisi de laisser l'ordinateur jouer sur une case aléatoirement.

Ce projet a été développé en JavaFX 2 dans l'objectif de fournir un jeu au projet **GameLoader**. Nous avons également utilisé JavaFX Scene Builder pour construire notre interface graphique via un fichier FXML.

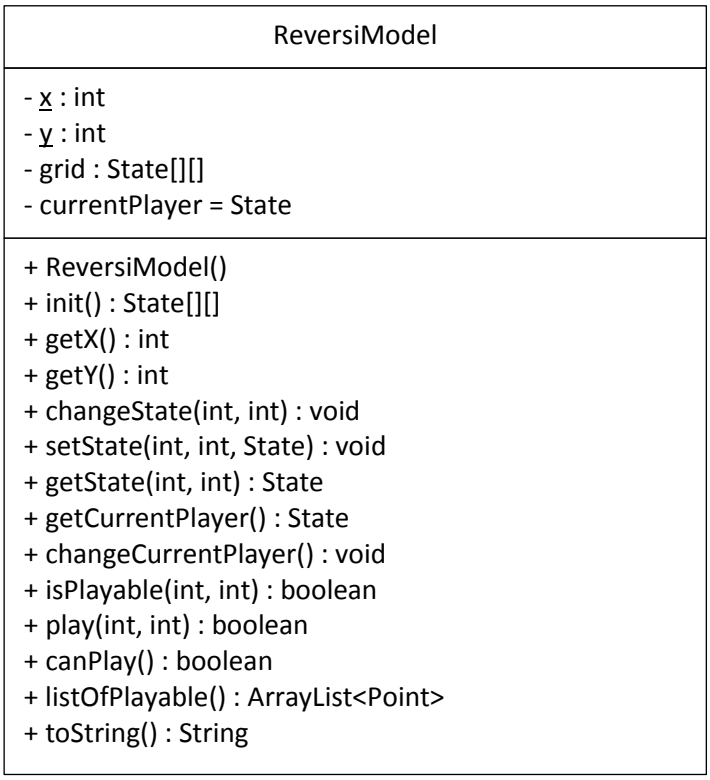
Ce que nous avons fait :

Nous avons 4 classes dans ce projet : *Reversi*, *ReversiModel*, *RulesController* et *SampleController* ; ainsi que 2 fichiers FXML : *Rules* et *Sample*. *Reversi* est la classe principale du projet, qui permet le lancement de l'application et de charger quelques ressources. *ReversiModel* est le modèle du jeu, qui permet de modéliser la partie, et de savoir où le joueur peut jouer suivant sa couleur, ainsi que d'effectuer l'action de jouer en retournant les pions nécessaires. Le jeu est modélisé par un tableau 8*8 de *State*. Voilà le code de l'énumération *State* :

```
public enum State {
    EMPTY("empty"),
    WHITE("white"),
    BLACK("black");

    private final String string;

    State(String string) {
      this.string = string;
    }

    @Override
    public String toString() {
      return this.string;
    }
  }
```

Retourner un pion signifie donc changer l'état d'une case du tableau.

Voilà le diagramme de classe de *ReversiModel* :

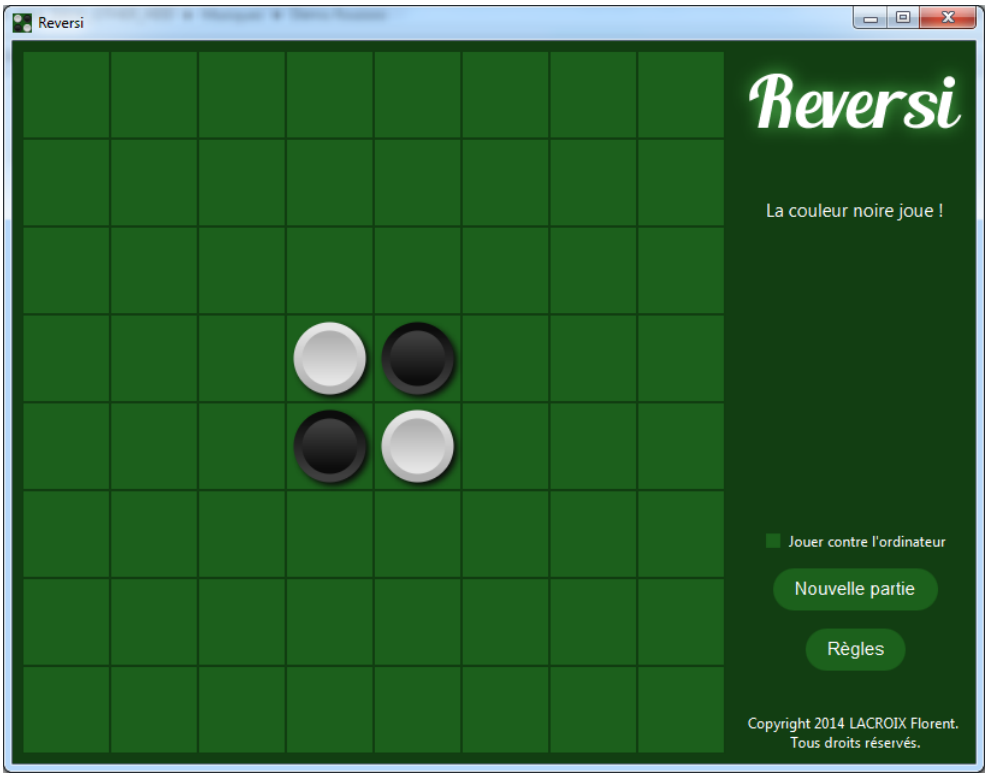


isPlayable() permet de savoir si le joueur courant peut jouer sur la case de coordonnées (x,y). play() permet au joueur courant de jouer sur une case (x,y) s'il le peut. canPlay() permet de savoir si le joueur courant peut jouer ce tour ci. listOfPlayable() retourne une *ArrayList* de *Point* qui indique où le joueur courant peut jouer. toString() fournit une représentation du plateau de jeu. Les autres méthodes sont triviales.

SampleController est la classe associée à *Sample* : c'est le contrôleur de la vue représentée par le fichier FXML. Elle représente le plateau de jeu, ainsi qu'une petite interface à droite permettant d'accéder à quelques informations durant la partie, de démarrer une nouvelle partie ou de consulter les règles du jeu.

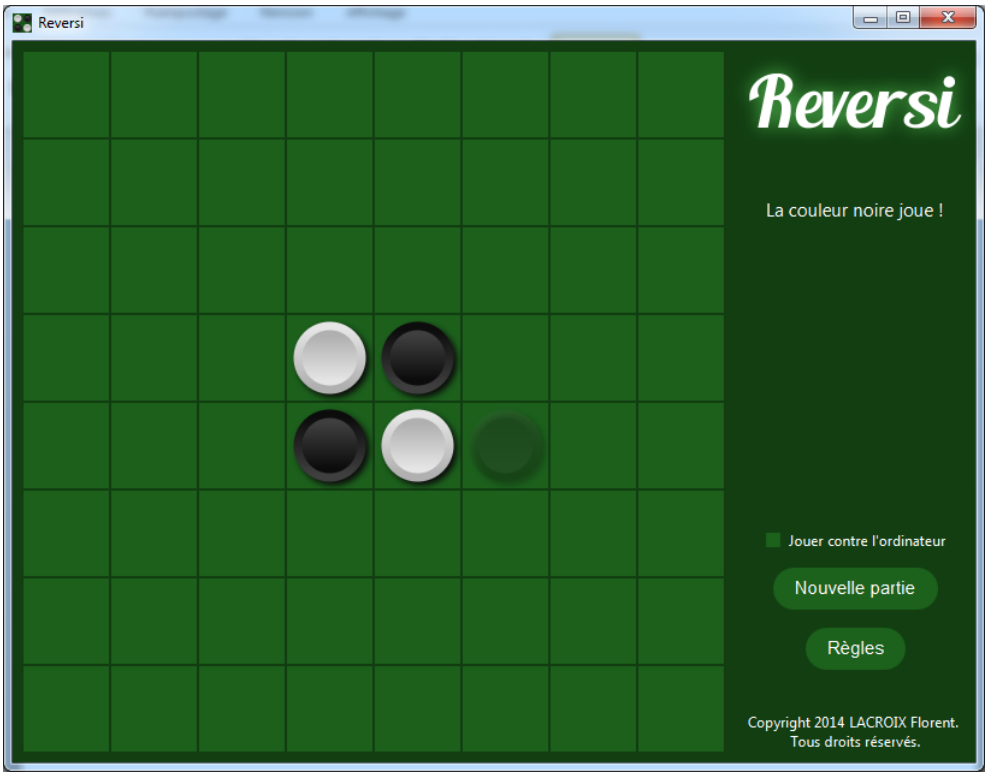
RulesController est la classe associée à *Rules* : c'est le contrôleur de la vue représentée par le fichier FXML. Elle permet d'afficher les règles du jeu.

Voici donc un aperçu de l'application quand on la démarre :

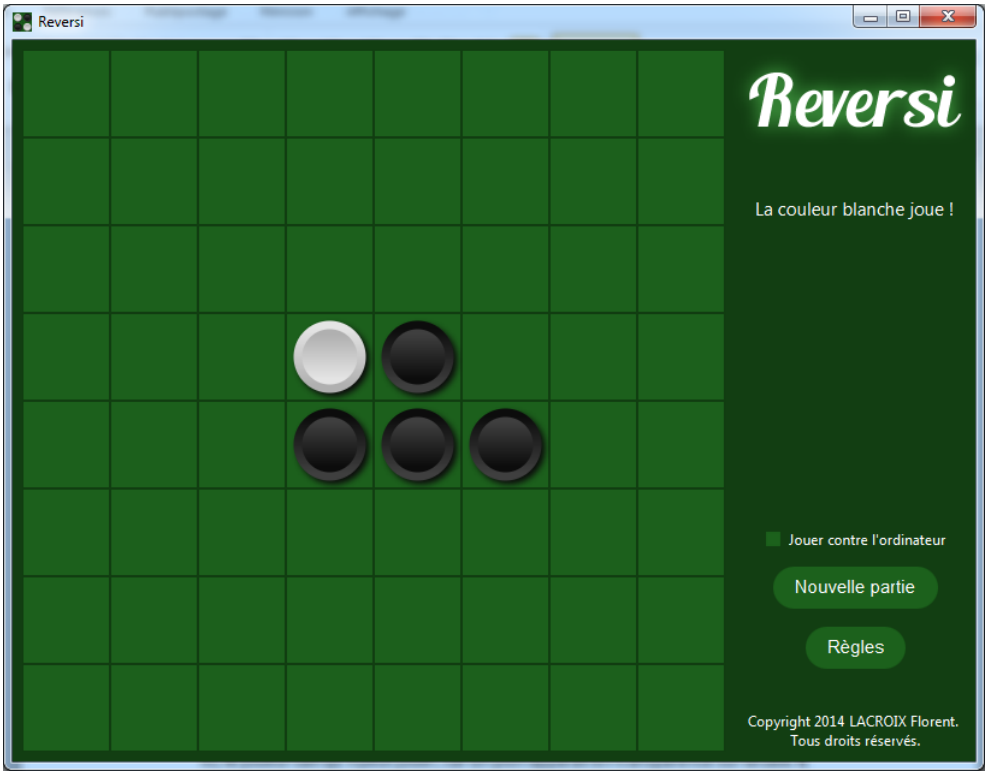


En cochant la CheckBox 'Jouer contre l'ordinateur', et en cliquant ensuite sur le bouton 'Nouvelle partie', on démarre une nouvelle partie contre l'ordinateur. Nous jouons alors la couleur noire.

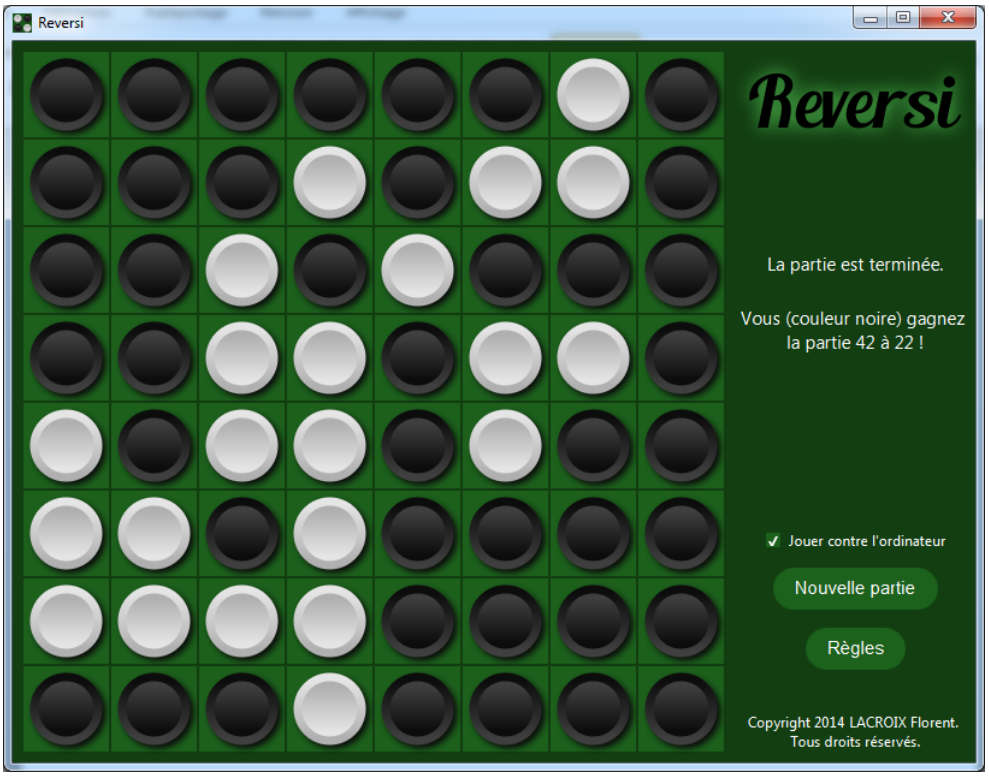
Le joueur qui doit jouer connaît les cases sur lesquelles il peut jouer car un pion apparaît en transparence. Il ne connaît en revanche pas (sans réfléchir du moins) l'intégralité des pions qui vont se retourner. En cliquant sur la case, certains pions vont se retourner, et la main passe à l'adversaire. Si celui-ci ne peut pas jouer, la main repasse automatiquement à l'autre joueur. Si aucun des deux joueurs ne peut jouer, la partie se termine et les scores s'affichent. Les messages sont bien évidemment personnalisés selon que l'on joue contre l'ordinateur ou contre un autre joueur. Le 'Reversi' en haut à droite prend alors la couleur du joueur gagnant, et redevient blanc au lancement d'une nouvelle partie. Cela donne les aperçus suivants :



Ici, le joueur sait qu'il peut jouer, car un pion apparaît en transparence sur la case où il a la souris.



Le joueur noir vient de jouer sur une case, les pions se retournent et la main passe à l'adversaire. Le message en haut à gauche a bien évidemment changé.



Ici, le résultat d'une partie contre l'ordinateur, que nous venons de gagner. Le 'Reversi' a bien changé de couleur. Nous pouvons redémarrer une nouvelle partie en utilisant le bouton prévu à cet effet.

On peut également consulter les règles en cours de partie, sans pour autant devoir recommencer une nouvelle partie :

Les règles apparaissent avec un fond légèrement transparent, laissant apparaître le jeu en cours.

L'utilisation de fichier FXML permet de facilement coder un modèle **MVC** : en effet, nous avons ici *ReversiModel*, le **M** ; *Sample*, le **V** ; et enfin *SampleController*, le **C**.

Un fichier CSS gère les graphismes (couleur, police, effet sur les boutons, barres de défilement,…). Les polices utilisées dans l'application, si elles ne sont pas standards (comme celles utilisées ici : Blenda Script) sont chargées par l'application, afin d'être sûr que l'utilisateur puisse en profiter.

Conclusion et améliorations possibles :

Il y a un petit souci visiblement au niveau des polices : en effet, nous voulions au départ avoir le 'Reversi' en haut à droite qui change de couleur pour représenter la couleur du joueur courant. Mais cela prenait à chaque fois un petit laps de temps qui rendait l'effet désagréable. Nous avons le même souci quand il s'agit d'afficher les règles du jeu. Nous n'avons pas encore réussi à le corriger (s'il est corrigeable). Une intelligence artificielle plus développée est également envisageable, mais nous ne sommes pas des joueurs très réguliers de Reversi, et ne connaissons donc pas les stratégies permettant de gagner. Nous voulions également ajouter quelques effets de transitions, mais nous n'y sommes pas parvenus (nous utilisions pourtant un code similaire à celui employé dans d'autres applications. Le blocage provient peut être du fait que les éléments sur lesquels nous tentions d'appliquer des effets de transitions étaient issus du FXML).

Néanmoins, le jeu est totalement opérationnel ; de plus les graphismes et l'ergonomie sont agréables pour l'utilisateur. Le cahier des charges est donc rempli.

Ce rapport succinct ne présente pas tout ce qui a été fait dans l'application, mais les aspects principaux, à savoir comment nous avons procédé, ainsi que quelques points de l'interface graphique qui nous ont paru intéressants.