Objectives: learning to manage Layout containers and to use them to play the Reversi game.

1 Reversi rules

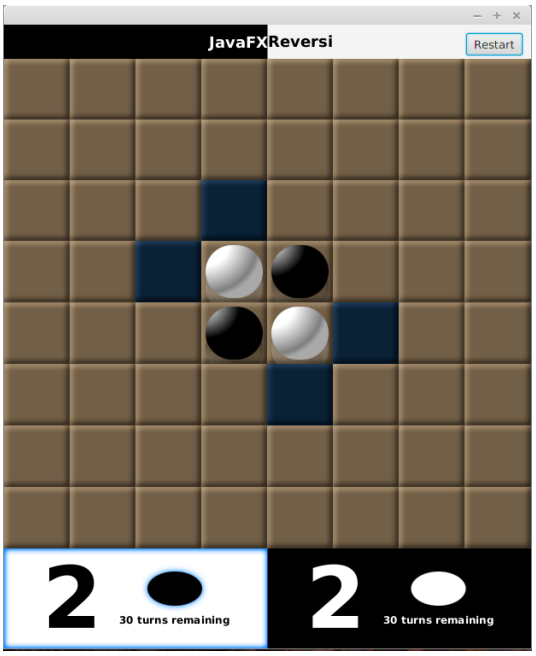

Figure 1: Initial Reversi board

You have to enclose same-colored lines with opponent pieces, in order to flip the color of every surrounded piece. The player who has the most pieces after a given number of turns wins.

The "Black" player begins. He can put down a black piece on any square adjacent to a white piece, as long as it closes a line bordering white-only pieces. Lines can be either horizontal, vertical or diagonal. As shown on fig. 1, blue squares indicate where pieces (either black or white, depending on the player) can be put down on the checkboard.

After the black piece has been put down, every white piece of the line surrounded by black piece turns black (the move can not be cancelled). The counter of black pieces is updated and the "White" player can play next. Fig. 2 shows the first moves of a game.

2 Game algorithmics

1. Download the `Project_A` Netbeans project.
2. Open the project and compile both `Owner.java` and `ReversiModel1.java` files.
3. Skim over `ReversiModel.java` to acquire an elementaty understanding of the methods.

3 Centering objects using StackPane

The StackPane layout is designed to stack objects in back-to-front order.

1. Compile and run `CenterUsingStack.java`(fig. 3).
2. What does happen if the order of stacked elements is reversed?
3. `StackPane`'s contents are centered. How to modify this behaviour?

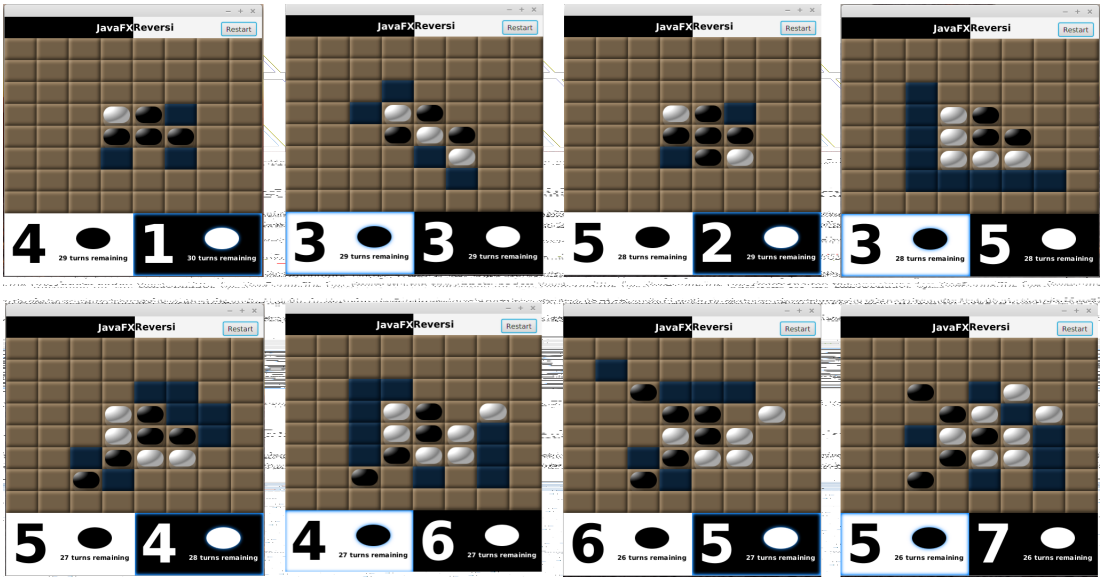

Figure 2: The game begins

Figure 3: Centering using `StackPane`

4 TilePane

The `TilePane` layout contains elements of same dimensions. Space is automatically partitionned in regions with the same surface area whic contain `TilePane`'s children.

The numbers of lines and columns can be given as parameters of the TilePane's constructor: Both width and height are dynamically adjusted. Conversely, both the numbers of lines and columns can be inferred from width and height.

1. Download the `Project_C` Netbeans project.
2. Compile and run `AlignUsingStackAndTile.java` (fig. ??).
3. What is the type of the `TilePane`'s children?
4. Note that both children keep equal dimensions regardless of the layout resizing.
5. Change the code to put each text in the middle of the associated tile.

Figure 4: Align with `TilePane`

5 FlowPane and Boxes

`HBox`, `VBox` are designed to group some objects along either a vertical (`VBox`) or horizontal (`HBox`) line. with or without separation. `FlowPanes` allow to "cut" the line when some limit has been reached.

1. Download the `Project_D` Netbeans project.
2. Compile and run `PlayerScoreExample.java` (fig. 5).
3. What is the purpose of the `DropShadow` class? Change the color and observe the effects. Same questions for the `InnerShadow` class.
4. Note how the generic `Region` class is used to manage backgrounds.
5. Identify the *CSS*-formatted code, used to update the `style` of the `background Region`.

Figure 5: Playing with `FlowPane` and `Boxes`

6 BorderPane

This layout allows to place children at border locations: top, bottom, left, right and center. It is advised to use those predefined positions instead of modifying the list of `BorderPane` children.

1. Download the `Project_E` Netbeans project.
2. Compile and run `BorderLayoutExample.java` (fig. 6).
3. Notice how children are positioned using `setTop()`, `setCenter()` and `setBottom()` methods.

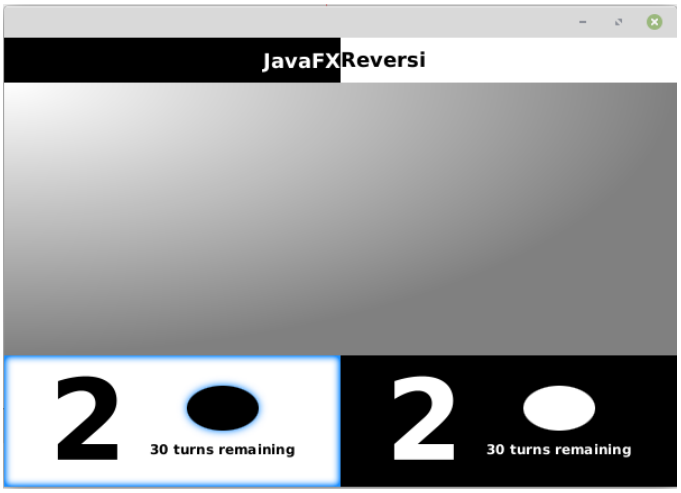

Figure 6: `Reversi` global `BorderPane` layout

7 Customized Regions

The `Region` previously seen is a generic class for any *JavaFX* node, intended to support *CSS* code.

This class allows to build dynamic containers "from scratch" and to combine them into a layout. Its main properties are

- `width`, `height`: refer to its dimensions as soon as the children layout is set up or redefined.
- `minWidth`, `maxWidth`, `prefWidth`, `minHeight`, `maxHeight`, `prefHeight`: minimal/preferred/maximal sizes. This data can be overidden by `Objects` subclassing `Region`.
- `padding`: amount of space surrounding the `Region`.
- `snapToPixel`: round both position and dimensions to integer values.

Download the `Project_F` Netbeans project.

7.1 Single board square

1. Compile and run `ReversiSquareTest.java` (fig. 7).
2. Notice how the `ReversiSquare` class's `Style` is defined.
3. Change the properties of both `Light` and `Lighting` objects to get various renderings.

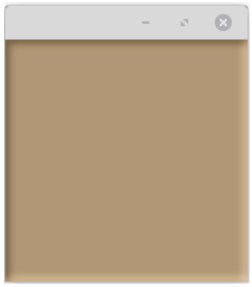

Figure 7: One gameboard Square

7.2 Single pieces

1. Compile and run `ReversiPieceTest.java` (fig. 8).
2. In `ReversiPiece.java`, why the piece's `radius` is equal to 0 when the related Square has no Owner?
3. Some "reflection effect" can be observed under the white and black discs. Modity the parameters of the `Reflection` object to get different results.

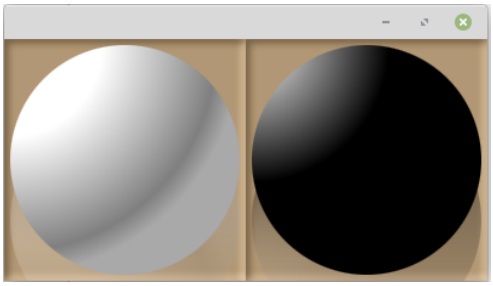

Figure 8: A pair of white and black pieces

8 GridPane layout

This layout defines a spreadsheet-like grid of "cells". Each cell is associated with a column number and a row number, and contains some children. Cells can be horizontally or verticallly merged.

Its main properties are

- `halignment`, `valignment`, `margin`;
- `hgrow`, `vgrow` (values: `NEVER`, `SOMETIMES`, `ALWAYS`) to override the preferred size; `fillWidth` et `fillHeight` to adjust the dimensions of a node to rows and columns.

1. Download the `Project_G` Netbeans project.
2. Compile and run `ReversiGridPane.java` (fig. 9).
3. The pieces are (on purpose) rendered as oval shapes insted of circular ones. How to modify the code in order to get circular shapes?
4. Notice how the `GridPane`'s nodes are defined by looping over the elements of `ReversiModel`.

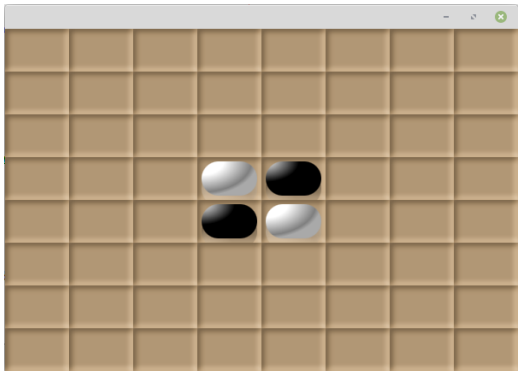

Figure 9: The complete boardgame

9 AnchorPane

This layout allows to position a node child to some fixed distance from the container's sides (fig. 10). A node can also be stretched either horizontally or vertically to follow side constraints.

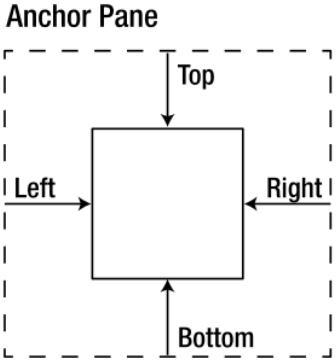

Figure 10: `AnchorPane`

The `AnchorPane` layout displays its children sorted in *First In, First Out* order. If no link is set between children and border sides, the children are located on the top-left corner.

1. Download the `Project_H` Netbeans project.

2. About `ReversiSquareFinal.java`:

   - What is the purpose of the `highlightTransition` object?
   - Note the call to the overridden `layoutChildren()` method. This method is very useful to control children rendering.

3. Compile and run `ReversiAnchorPane.java`. Notice in particular how positioning constraints are defined on `game` and `restart` variables.

4. Enjoy the game!