

Notes on Assignment 5

Samir Genaim

Objectives

In this assignment we have two objectives

1. Add support for the Reversi Game, following the design that we have introduced in the previous assignments -- just to "appreciate" the good design we had!
2. Allow the main window to have an automatic player -- to practice threads and interrupts

Reversi

- ◆ You **MUST** implement the Reversi game following the design we used in the previous assignments, namely
 1. Implement ReversiFactory
 2. Implement ReversiRules
 3. Implement ReversiMove
 4. Implement Random and Console Player
 5. Integrate it in both console and window mode
 6. etc.

Automatic Player

- ✦ The automatic player will be some code that sleeps for some time, and then makes a random move. It is just to simulate a process that takes some time so we can practice threads with swing
- ✦ The “difficult” part is how to make the automatic player play automatically when its turn comes, without any user intervention
- ✦ We will see one way to do it, that might be the simplest given the overall design that we have followed in the 3rd assignment
- ✦ Most of the work will be in the controller

1st - PlayerType aware Counters

Let us first make the counters aware of the fact that they are played by human or automatic. This is not compulsory, but it simplifies things a bit:

```
public enum Counter {  
    EMPTY("Empty"), WHITE("White"), BLACK("Black");  
  
    String name;  
    PlayerType mode = PlayerType.HUMAN;  
  
    Counter(String name) {  
        this.name = name;  
    }  
  
    public PlayerType getMode() {  
        return mode;  
    }  
  
    public void setMode(PlayerType mode) {  
        this.mode = mode;  
    }  
  
    public String toString() {return name;}  
}
```

```
public enum PlayerType {  
    HUMAN("Human"), AUTO("Automatic");  
  
    private String name;  
    PlayerType(String name) {  
        this.name = name;  
    }  
  
    public String toString() {  
        return name;  
    }  
}
```

2nd - ComBoxes for Player Types

Add the ComBoxes to the control panel, so the user can select the type of player

```
whitePlayerList = new JComboBox<PlayerType>(new PlayersModel(Counter.WHITE));  
blackPlayerList = new JComboBox<PlayerType>(new PlayersModel(Counter.BLACK));  
...
```

All the work of setting the player will be done in the class **PlayersModel**, which is a class that implements the interface **ComboBoxModel<PlayerType>**.

We assume that **PlayersModel** has an access to the **controller** (either define it as an inner class or pass it the controller in the constructor)

YOU MUST DO IT THIS WAY SO
YOU PRACTICE SWING'S MODELS

3rd - PlayerModel

```
public class PlayersModel implements ComboBoxModel<PlayerType> {  
    private Counter player;  
    private PlayerType selected;  
  
    public PlayersModel(Counter player) {  
        this.player = player;  
        this.selected = player.getMode();  
    }  
  
    public int getSize() {  
        return 2;  
    }  
  
    public PlayerType getElementAt(int index) {  
        ...  
    }  
  
    public void setSelectedItem(Object anItem) {  
        this.selected = (PlayerType) anItem;  
        ctrl.setPlayerMode(player, this.selected);  
    }  
  
    public Object getSelectedItem() {  
        ...  
    }  
    ...  
}
```

1. When the user selects an item, this method is called with the selected item, i.e., with `PlayerType.HUMAN` or `PlayerType.AUTO`
2. We tell the controller that the player has changed, later we see what `setPlayerMode` does

4th: Ignore user Clicks

In the current implementation, when the user clicks on a position on the board we execute something like this in the corresponding listener

```
if ( isActive ) {  
    ...  
    ctrl.makeMove(col,row,turn);  
}
```

Now we should check also that the current player is human, i.e., check what is the mode of `turn`. This way we ignore user clicks when an automatic player is playing. Another possibility is to disable the board if you are using a grid of JButton (but it is not the best choice)

We are done with the view so far, we will mention some more requirements at the end. Let us modify the controller to support the automatic player

```
public class WindowControler extends Controller {
```

```
    // FROM PREVIOUS ASSIGNMENT (WILL BE MODIFIED A BIT)
```

```
    public void run() { ... }
```

```
    public void makeMove(int col, int row, Counter turn) { ... }
```

```
    public void undo() { ... }
```

```
    public void reset() { ... }
```

```
    public void changeGame(GameType gameType, int dimX, int dimY) { ... }
```

```
    public void randomMove(Counter player) { ... }
```

```
    public void requestQuit() {...}
```

```
    // NEW METHODS
```

```
    private void stopAutoPlayer() { ... }
```



```
    private void automaticMove() { ... }
```

```
    public void setPlayerMode(Counter player, PlayerType mode) { ... }
```

```
}
```

5th: The Automatic Player

We define a method to start a thread for the automatic player, this method will be called from several places where the turn might change to an automatic player

```
private void automaticMove() {  
    if (game.getTurn().getMode() == PlayerType.HUMAN)   
        return;  
  
    autoThread = new Thread() {   
        public void run() {  
            ...  
            while (next turn is AUTO && game not finished && thread not interrupted) {  
                1. sleep for a while  
                2. if the thread was not interrupted, execute a random move  
            }  
            ...  
        }  
    };  
    ...  
}
```

Does nothing if the current player is not automatic

Declare as field, so we can interrupt the thread later

The while loop permits handling the case in which both players are automatic

6th: When to call automaticMove

Call it whenever the turn can change, if it does not correspond to an automatic player then automaticMove does nothing (see first line of automaticMove), otherwise it will start the corresponding thread, etc. Here are some places where it should be called, there are more places that you should find by yourself

```
public void undo() {  
    ...  
    automaticMove();  
}
```

```
public void makeMove(int col, int row, Counter turn) {  
    ...  
    automaticMove();  
}
```

```
public void run() {  
    ...  
    automaticMove();  
}
```

```
public void setPlayerMode(Counter player, PlayerType mode) {  
    player.setMode(mode);  
    automaticMove();  
}
```

7th: Stopping the Automatic Player

While the automatic player is playing, we want to be able to reset or change the game. This means that we have to stop the thread of the automatic player first

```
private void stopAutoPlayer() {  
    if (autoThread != null) {  
        // 1. interrupt autoThread  
        // 2. wait for autoThread  
           to terminate  
    }  
}
```

It is very important to wait for the thread to finish!! To make sure that we execute actions in the right order

```
public void reset() {  
    stopAutoPlayer();  
    ....  
}
```

8th: some more requirements

- ◆ The "Undo" and "Random Play" buttons should be active only when the HUMAN player is playing
- ◆ If we undo a move of an automatic player, it should play again -- i.e, we will never be able to undo more than two consecutive moves in such case
- ◆ You cannot make direct calls from the controller class to anything in the view Classes -- disable/enable buttons should be done in the GameObserver methods
- ◆ All modifications to swing components that might not execute in swing's dispatching thread MUST be done using `SwingUtilities.invokeLater`

9th: Optional

- ♦ Add command line parameters to specify the type of players (in the window mode), e.g.,
`--black human --white auto`
- ♦ Make the automatic player intelligent, using the MinMax algorithm
- ♦ Be creative ...