Java Programming 2 – Lab Sheet 7

This Lab Sheet contains material based on the lectures up to and including the material on GUIs and Swing.

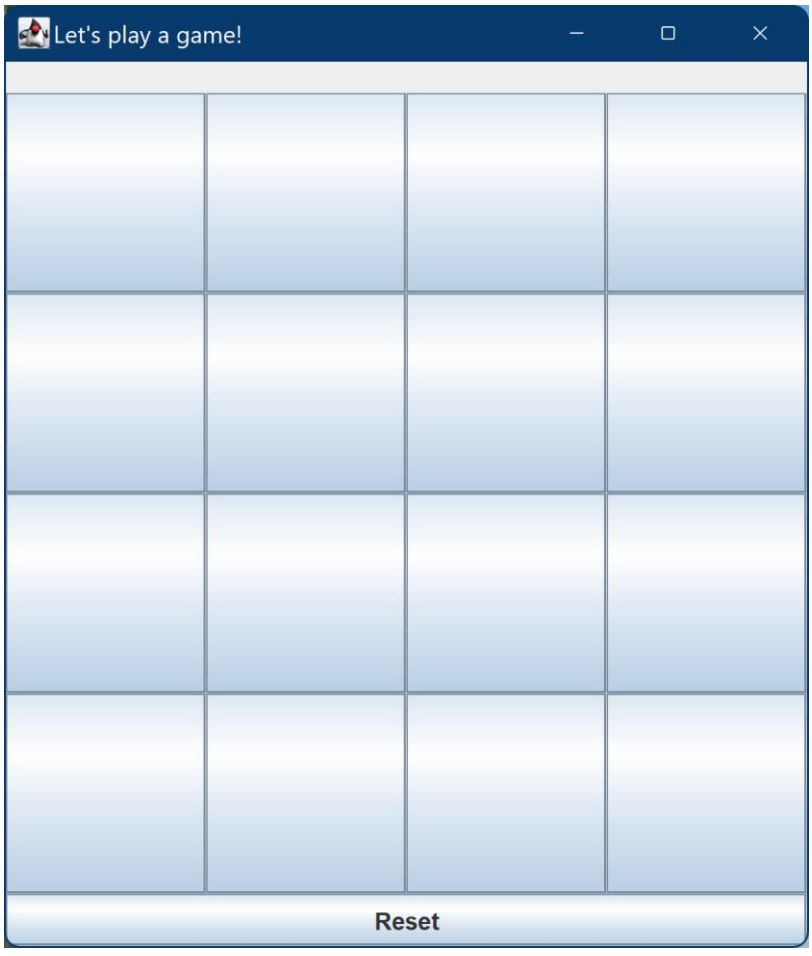
The deadline for Moodle submission of this lab exercise is 12:00 noon on Friday 12 November 2021.

Aims and objectives

- Writing callback methods for a Swing GUI
- Designing Java classes and methods based on a less detailed specification

Background

The code you have been given in the **GameWindow.java** file is the start of a simple GUI-based game. If you run the main method of **GameWindow** (don't forget that it is in the **game** package), you will see a window pop up similar in appearance to the following:



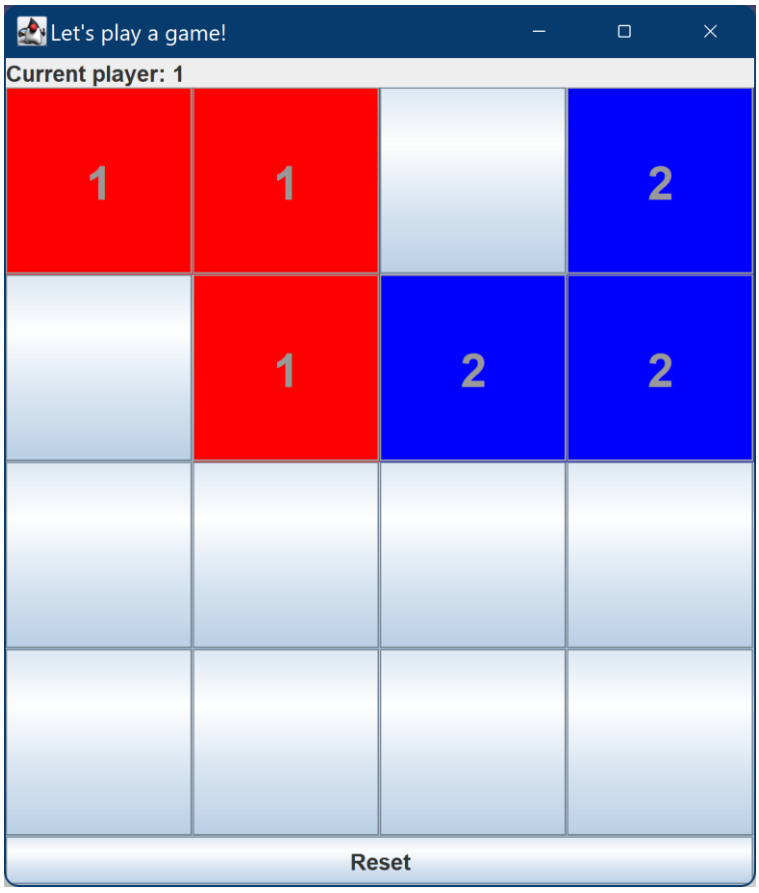
When you click the buttons on this window, nothing happens for the moment. Your task is to write the back-end methods and classes to turn this window into a GUI-based game.

The rules of the game are as follows:

- There are two players, Player 1 and Player 2.
- The players take turns making a move until either one of them wins the game, or else the board is filled up without either player winning.
- On each turn, a player clicks on one of the available buttons. That button should change its background colour to indicate that it has been selected, and should also change its text to indicate which player has chosen it (either “1” or “2”)
- Once a button has been clicked by one of the players, it is unavailable to be clicked any more.
- A player wins if they select all buttons in a 2x2 square.

Note that both Player 1 and Player 2 work on the same, shared game board: in a real game-playing session, the players would need to pass the computer mouse back and forth between each other.

For example, the board might resemble the following after a few turns of play. Note that the red and blue squares are not clickable; the only squares that respond to a click are the grey squares that have not yet been claimed.



As shown by the text at the top of the window, it is Player 1’s turn to make a move. If Player 1 clicks on the open square in the second row, they will win the game by creating a 2x2 red square. If they click anywhere else, Player 2 will have the opportunity to make another move. This process continues until either one of the players wins, or else the board is completely filled with neither player winning.

The “Reset” button at the bottom can be used at any time to restore the grid to its original state, with no squares claimed – i.e., the board should be returned to the appearance shown on the first page of the lab sheet.

The accompanying video (on Moodle) gives a full demonstration of the intended behaviour.

Submission material

Your task is to update the provided classes to implement the game behaviour as described above. Note that the provided **GameWindow** class is in the **game** package, and your newly created **GameButton** class should also be in the same package.

GameButton class

The first task is to create a subclass of **javax.swing.JButton**, called **game.GameButton**, which will be used to implement the game behaviour. Implementing the necessary behaviour will require calling various methods of the parent **JButton** class, as suggested below. You can read the documentation of **JButton** at https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/javax/swing/JButton.html.

Your **GameButton** class should provide the following methods:

- **public void setSymbol(String symbol)**
  - Sets the symbol to be shown on the button (for this game, the symbol will be “1” or “2”). In addition to storing the symbol, this method should also update the properties of the button as follows:
    - Change the text of the button to show the symbol (use **setText()**)
    - Change the background colour of the button – you can choose which colour to use for each player, but be sure that the two colours are different. In the example we used red and blue. You can change the background using **setBackground()**, which makes use of the **java.awt.Color** class.[1]
    - Configure the button so that it cannot be clicked any more using the **setEnabled()** method
- **public String getSymbol()**
  - Returns the current symbol set on this button. If no symbol has been set, this method should return **null**.
- **public void reset()**
  - This method should reset the **GameButton** back to its initial state (i.e., it should undo all of the changes made in **setSymbol**). Specifically:
    - Change the text of the button back to the empty string and remove any stored symbol (i.e., **getSymbol()** should return **null** after **reset()**)
    - Change the background colour back to the default – you can use **setBackground(null)** to do this
    - Re-enable the button using **setEnabled()**

[1] Note that the **java.awt.Color** class uses the American spelling of “color”.

Changes to GameWindow

Once you have created your **GameButton** class, you should make the necessary changes to **GameWindow** to support the game. This involves the following high-level steps; it is up to you to work out the details of how to make this work.

1. Add a mechanism for keeping track of the current player
2. Replace the array of **JButton**s with an array of **GameButton**s
3. Update the **GameWindow** class to implement the **ActionListener** interface
4. Add the **GameWindow** instance (i.e., **this**) as an action listener to all of the **GameButton**s, as well as to the **resetButton**.
5. Implement an **actionPerformed()** method in **GameWindow** that responds to a click on any of the buttons to support the game play. The suggested behaviour of **actionPerformed()** is given below.

GameWindow.actionPerformed() suggestions

In your **actionPerformed()** method, you can check which button was clicked on by calling **getSource()** on the **ActionEvent** parameter.

If the user clicked on the "Reset" button, you should reset the game board back to its initial configuration and make no other changes.

If the user clicked on any of the **GameButton** instances, you should update the state of that button to reflect the user that clicked on it, using the appropriate method of **GameButton**.

After every **GameButton** update, you should also check through the grid of buttons to see if either of the game-ending conditions holds:

- If one of the players has won, update the content of **statusLabel** to say "Winner: Player *n*" (where *n* is either 1 or 2). In this case, all **GameButton**s should be disabled with **setEnabled()** so that the players do not continue to click buttons after the game is over.
- If the board is full with no winner, update the content of **statusLabel** to be "Board is full with no winner"

If the neither of the game-ending conditions holds, you should switch to the other player and update the **statusLabel** to say which player is the next one, and the game will continue.

Hints and tips

1. Please watch the provided video carefully to see the intended behaviour in action.
2. You can add any number of **private** helper methods to **GameWindow** to support the required behaviour – in fact, you are suggested to do this! – but make sure that you do not add any **public** methods or fields.
3. The specification for this lab is deliberately more open-ended than in previous labs, meaning that you are free to use any implementation technique you want, particularly for the modifications to **GameWindow**. If you are not sure how to approach the lab, please talk to your tutor or demonstrator for suggestions.

How to submit

You should submit your work before the deadline no matter whether the programs are fully working or not. Before submission, make sure that your code is properly formatted (e.g., by using **Ctrl-Shift-F** to clean up the formatting), and also double check that your use of variable names, comments, etc is appropriate. **Do not forget to remove any "Put your code here" or "TODO" comments, and also remove any debugging println statements!**

When you are ready to submit, go to the JP2 Moodle site. Click on **Laboratory 7 Submission**. Click 'Add Submission'. Browse to the folder that contains your Java source code – probably **.../eclipse-workspace/Lab7/src/game/** -- and drag only the *two* Java files **GameButton.java** and **GameWindow.java** into the drag-and-drop area on the moodle submission page. **Your markers only want to read your java files, not your class files.** Then click the blue save changes button. Check the files are uploaded to the system. Then click **submit assignment** and fill in the non-plagiarism declaration. Your tutor will inspect your files and return feedback to you via Moodle.

Outline Mark Scheme

Your tutor will mark your work and return you a score in the range "Excellent" (*****) to "Very poor" (*). We will automatically execute your submitted code and check its output; we will also look at the code before choosing a final mark.

Example scores might be:

5*: you complete the code correctly with no bugs and correct style

4*: you compute the class mostly correctly with minor bugs OR your code is correct but the style is inappropriate

3*: more major bugs and/or more major style issues

2*: some attempt made

1*: minimal effort

For this assignment (and all future ones), we will also be considering code style – comments, formatting, variable names, etc – in addition to correctness. You will be deducted one star if your code style is not appropriate. Please check with the tutors and demonstrators during your lab session if you are uncertain about style.

Possible extensions

If you have completed the core tasks of the lab and want to try something extra, here are some things to try. Please do not submit any code for these tasks; your submitted code should meet only the specifications above.

1. Try implementing other winning conditions – for example, can you implement a version of Tic-Tac-Toe, Connect Four, or Battleship?
2. Try reconfiguring the game so that Player 1 and Player 2 each have their own board to click on.
3. Try extending the game to support multiple players (more than 2).