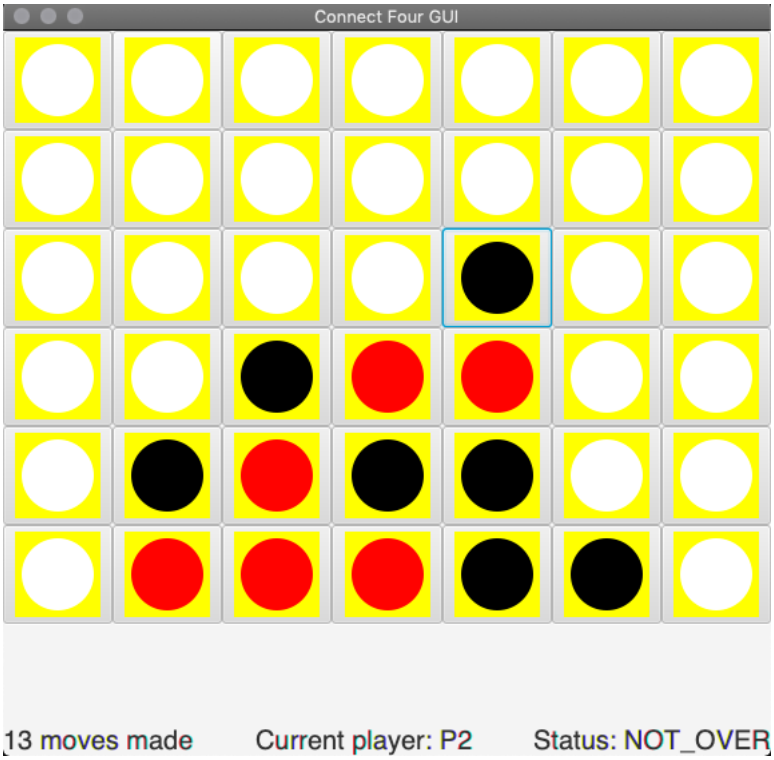
CSCI-605 Advanced Object-Oriented Programming Concepts

Homework 8: Connect Four GUI

03/28/2021



1 Introduction

This week we learned about the Model-View-Controller (MVC) architecture pattern and how it applies to GUI development.

In this homework we will look at the popular two player board game, Connect Four. Each turn alternates with the current player dropping a disc of their color into a column of the board. The winner is the first player to connect four of their discs together either horizontally or vertically.

You will be given a plain text UI version of the homework that adheres to the MVC architecture, and you will be tasked to implement an equally compliant JavaFX GUI version.

1.1 Goals

Students will gain experience with the JavaFX fundamentals such as:

- JavaFX Hierarchy
- JavaFX UI Controls
- JavaFX Layout Components
- MVC pattern

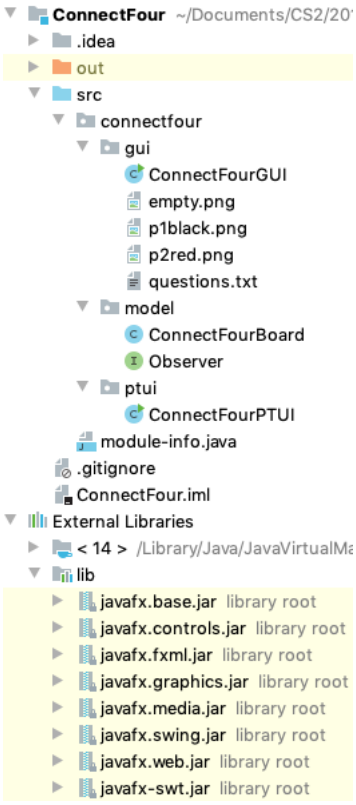
1.2 Provided Files

1. The **src** contains a plain text UI version of the game. `ConnectFourPTUI` is already completely written for you and has a run configuration for it (no command line arguments are used). Play the game by inputting column numbers (the first column is 0) so you are familiar with how it works.
2. The **sample_run.txt** file contains a full game play example of the PTUI.

2 Implementation

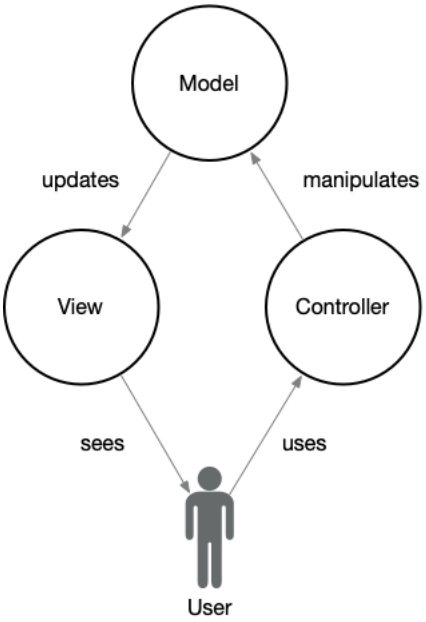
2.1 Project Structure

Your project should be structured as follows:



2.2 Design

Recall from lecture the overall design of the Model-View-Controller architecture.



2.2.1 Model

The model is responsible for managing the data and the rules of the application. It receives input from the user from the controller. When the state of the model changes, it notifies its view observers to update.

The complete model is given to you. It contains:

- `ConnectFourBoard`: The model class that contains the representation of the grid of discs, along with statistics and the status of the game.
- `Observer`: An interface the view implements so that it may receive updates from the model.

2.2.2 View

The view is the visual presentation of the model that the user sees. It registers itself as an observer of the model so it can receive and display the current state of the model. Using this design there can be different views that display things in a different format, e.g. textually vs graphically.

There are two views that serve as the client main programs for this application.

- `ConnectFourPTUI`: A fully functional plain text UI.
- `ConnectFourGUI`: The JavaFX compliant GUI that you will implement.

2.2.3 Controller

The controller is not a separate class here. Because it is so small, it is to be integrated into the view components. The PTUI has comments in the code to distinguish which piece is acting as which (look at the run method). When you develop your GUI, the code that responds to the button presses and informs the model is the controller part.

2.2.4 MVC with ConnectFourPTUI

Because the plain text UI is fully written and functional, you are encouraged to use as much code from it as you want to when writing your GUI.

2.3 Activity 1: MVC

Before you begin implementing your JavaFX GUI, `gui.ConnectFourGUI`, you should make sure you fully understand the MVC architecture that is provided to you. Make sure you read over the design section. Also, the code for ptui.ConnectFourPTUI is a great reference because it shows how as a view/controller it interacts with the model, `model.ConnectFourBoard`.

For this activity, you must answer the questions about the MVC architecture in the

`src/connectfour/gui/questions.txt` text file.

2.4 Activity 2: Implementation

2.4.1 GUI requirements

You are free to develop your GUI how you wish, but it must meet the following requirements:

- The title of your window should be ConnectFourGUI.
- Your window should contain:
  - An area where the grid of discs are displayed.
    - Clicking on any button in a column will cause a disc to be dropped into it (as long as the column is not full and the game is not over).
    - Images must be used for the placed discs. You are free to change the provided images but please keep them appropriate.
  - Three pieces of information should be displayed and updated automatically as the game is being played:
    - The number of moves made in the game, which corresponds to the number of discs played.
    - The current player to take the next turn.
    - The status of the game, which is NOT_OVER while the game is being played, and either P1_WINS, P2_WINS, or TIE when the game has ended.

    This information should not be editable by the user.
- When the game is over, the GUI should stay up in a disabled state until the user closes it.

2.4.2 Hints/Suggestions

- Please make certain that you do not use absolute paths to the images in your GUI! Look how the PokemonButtons demo from the lecture code does this:

```
/** bulbasaur image */
private Image bulbasaur = new Image(getClass()
                                    .getResourceAsStream("bulbasaur.png"));
```

All the images ride directly in the gui directory so you just need to specify the file name.

- All GUI classes should come from the `javafx` package. Beware, because your development tool may erroneously suggest a class with the same name from the `java.awt` or `javax.swing` packages.

3 Submission

You will need to submit all of your code to the MyCourses assignment before the due date. You must submit your src folder as a ZIP archive named "hw9.zip" (if you submit another format, such as 7-Zip, WinRAR, or Tar, you will not receive credit for this homework).

4 Grading

The following deductions will apply:

- Completeness/Implementation: up to 100% if a part of the writeup is not observed.
- Correctness/Testing: up to 50% or affected part if the solution is not correct.
- Quality/Style/Documentation: up to 50% if solution is poorly designed.
- Explanation: up to 100% if the student is not able to respond grader's questions.