CS 342 **Project#2** **Fall 2021**

Connect Four JavaFX GUI

Due Date:
Part #1 UML class diagram: Sunday, October 17th 2021, @11:59pm
Part #2 complete program: Sunday, October 24th 2021, @11:59pm

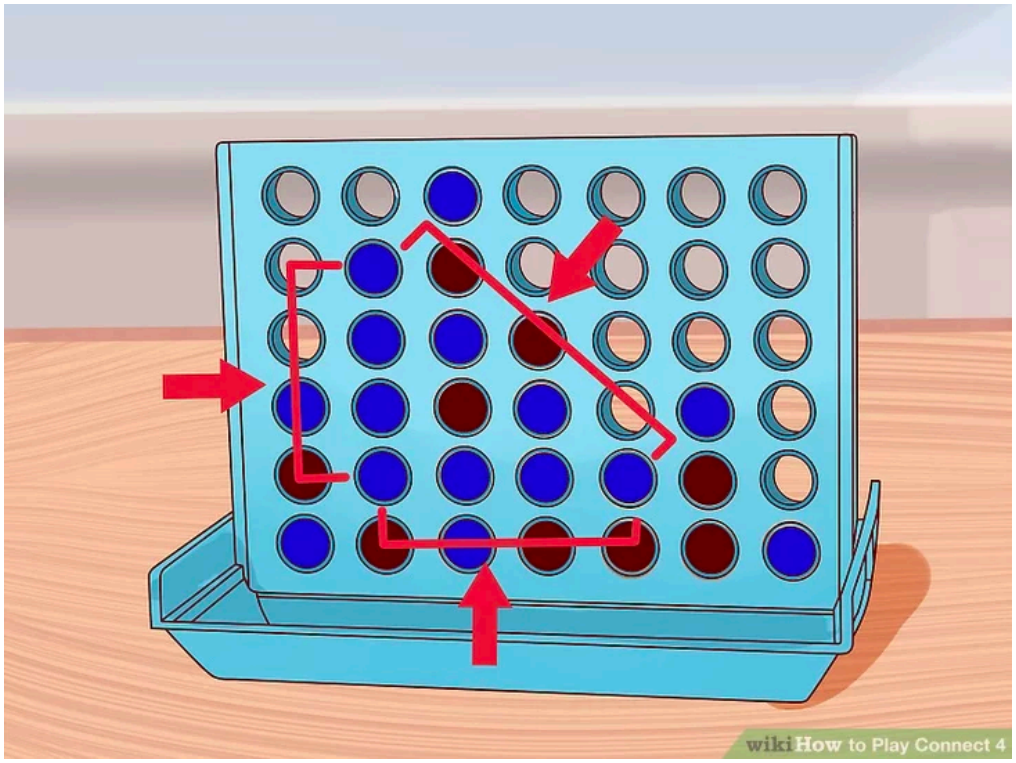
Description:

In this project you will implement the classic game of Connect Four. This is a somewhat simple game to understand and play which should allow you to focus on learning GUI development in JavaFX and trying your hand at event driven programing. You will code the user interface and all logic needed for clients to play your game.
This project will be developed as a Maven project using the Fall2021Project2_connect4 Maven project posted with this project description on our course BB site. **You may work in teams of two but do not have to.**

How the game is played:

Connect Four is played on a grid of 7 columns and 6 rows (see image above). It is a two player game where each player takes a turn dropping a checker into a slot (one of the columns) on the game board. That checker will fall down the column and either land on top of another checker or land on the bottom row.

To win the game, a player needs to get 4 of their checkers in a vertical, horizontal or diagonal row before the other player (see image below):



https://www.wikihow.com/Play-Connect-4

CS 342 **Project#2** **Fall 2021**

Your implementation of the game:

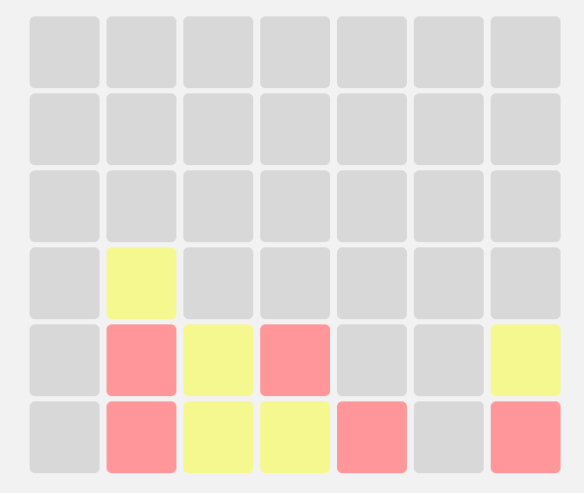
In your implementation of the game, the game board must be represented by the JavaFX component **GridPane**.

https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/GridPane.html

The checkers in your game will be from a class you create called **GameButton** which will extend the JavaFX class **Button.**
You will create a game board of **GameButton** instances that will populate your **GridPane.**
Moves will be made by clicking on a **GameButton** in your **GridPane** (see following images for an example)

This is the only way you are allowed to implement this. Failure to do so will result in a zero for this project.

GridPane with buttons



Some buttons clicked

CS 342 Project#2 Fall 2021

Implementation Details:

You may add as many other classes, data members, interfaces and methods as necessary to implement this program. You may **only use JavaFX components** for your GUI and they must be implemented programmatically in your code.

You may **NOT use Java Swing of Java AWT. You may NOT use FXML or Scene builder.**

You must follow the implementation here exactly. Failure to do so will result in a zero for this project!

The GUI:

You are welcome to use/discover any widget, pane, node, layout or other in JavaFX to implement your GUI in addition to the required elements described above. **Once again, for this project, you are not allowed to use Scene Builder or FXML layout files**.

The following elements are required:

1) Your program must start with a welcome screen that is it's own JavaFX scene. It will consist of:

Your welcome screen should welcome players to the game and have some sort of design other than the default color and style of JavaFX. In past semesters, this is a good opportunity to use images as background and play with the style of the graphical elements.

It will also have a button that allows the player to start playing the game. This will change the GUI to the game play screen.

2) The game play screen is its own JavaFX scene. It will consist of:

a) A menu bar with three menus: "*Game Play*", "*Themes*" and "*Options*"

<u>Game Play Menu:</u>
The Game Play menu will have a single menu item that is clickable called "*reverse move*"
clicking "*reverse move*" will undo the last move made and allow another move from that player. Clicking "*reverse move*" several times will undo the previous N moves.

For instance, lets say we have the following moves in order with p1 making the first move of the game:

p1, p2, p1, p2, p1, p2

Clicking "*reverse move*" three times would result in:

p1, p2, p1

With p2 making the next move

Themes menu:
This menu will have three menu items called "original theme", "theme one" and "theme two".

- Clicking on either "theme one" or "theme two" will change the look and feel of GUI. Changing colors, fonts, images, etc. It is up to you what themes you want to implement. You can think of it like a skin for your mobile device(holiday theme, pinball theme, Harry Potter theme, Star Wars theme….you get the idea).
- Clicking on "original theme " will return to the design when the game started.

Options menu:
This menu will have three items: "how to play", "new game" and "exit"

- Clicking on "how to play" will display some text on how to play the game.
- Clicking on "new game" will end the current game, reset the board and allow a new game to be played.
- Clicking on "exit" will exit and terminate the program.

b) an area displaying if it is player one or player two's turn to go

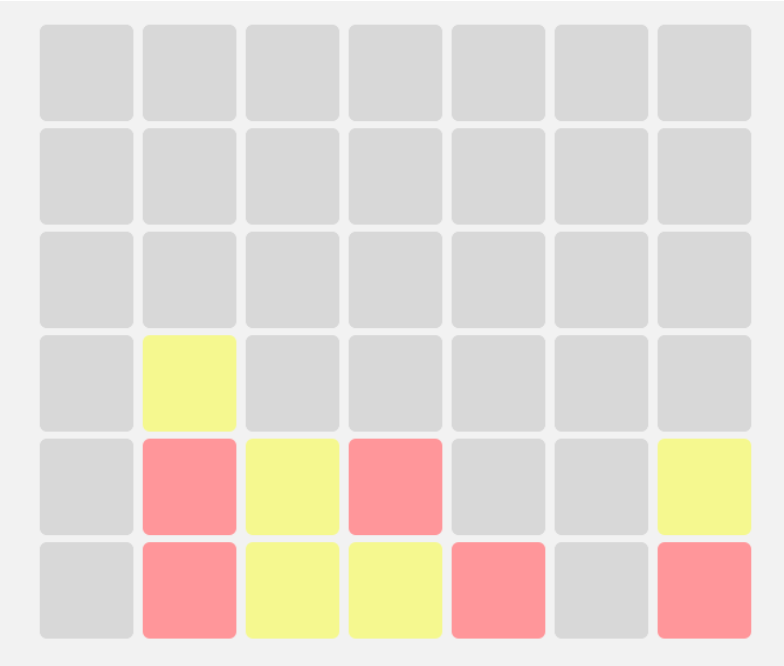
c) an area displaying each move made. It should have the following info for each move:

- which player made the move
- where the move was
- if it is a valid move

For instance, if player one attempted to make the first move of the game in row 3, column 3, you might print out to the GUI:

"Player one moved to 3,3. This is NOT a valid move. Player one pick again."

D) The game board as a GridPane filled with GameButtons:



- The buttons inside of the GridPane must be larger than the default size so your board is large enough to see and play.
- The buttons must start out as a different color or image than the default color. The image above starts them as “lightgrey”.
- Each players move must turn the button a unique color or image. The image above just turns the buttons yellow or red depending on whose turn it is
- The spacing of the buttons in the GridPane must be more than the default spacing. The image above has increased the spacing so one can “see” the grid.
- You may style the GridPane in any other way you think enhances the look and feel of the game.

3) A separate JavaFX Scene that is displayed when a player wins the game or there is a tie. This scene will have three elements:

- A message announcing who won the game or if there was a tie game
- An option to play another game(a button is fine for this)
- An option to exit the program (could also be a button)

CS 342 Project#2 Fall 2021

Playing the game in your Program:

Each player will take a turn choosing a spot on the GridPane. After clicking a GameButton to make a move, that button should change to the color or image that represents that player and become disabled so it can not be pressed again.

If the “reverse move” menu item is clicked, the last GameButton clicked should become enabled and switch back to it’s original color. It would then be that player’s, that made that move, turn again and this should be displayed in the area that shows who’s move it is.

In the event that there is a “connect four” either horizontally, vertically or diagonally, all remaining enabled GameButtons should be disabled and those GameButtons that make up the four connected spots should be highlighted (you can put text on them, change them a different color or anything that is noticeable so that the players can see it). Your game should pause for about 3-5 seconds and then switch to a third unique scene announcing who won and allowing the users to play again or exit the program. Buttons that restart the game or exit are fine for this scene.

If the user decides to play again, the game play scene should come up with everything reset and able to play again. If the user decides to exit, just terminate the program.

In case of a tie game, display the third scene and announce that the game ended in a tie. The user will then be able to play again or exit.

Testing Code:

You are required to include JUnit5 test cases for your program. Add these to the src/test/java directory of your Maven Project. While you will not be able to test buttons and user interaction with JUnit5, you can test methods that evaluate game play or implement game logic.

Do Not put your whole program in one method. Split things up into appropriate methods and classes.

CS 342 Project#2 Fall 2021

How to Start:

Some of you are used to just starting to code with no real plan for what you are doing. This project will be very painful with that approach. You must be systematic and thoughtfully plan out how various events will drive your program. You must also thoughtfully plan out how the user is allowed to interact with your program and how the user will know what to do next. I suggest the following:

- Draw out the user interface before you start to code. Decide what it looks like, what the JavaFX components are and how they are laid out in the window.
- From your drawing, decide what the user can interact with and how. What happens when the user say, clicks on a certain button?
- Next, decide what the EventHandlers should do and what your code needs to check every time the user interacts with your game.
- From your drawing, "play your game", write out what happens at each step, what you need to do programmatically at each step, what things do you need to keep track of, what things do you need to calculate.
- All of the above happen BEFORE you code one line. The better the plan, the more you think it through, the better it will go when you start to code and the fewer "unexpected" conditions that will pop up.

Electronic Submission:

If you worked in a group:

- only one of you needs to submit a project.
- You **must** submit a PDF file called Collaboration.pdf to the submission link on BB. In that document, put both of your names and netIds as well as a description of who worked on what in the project.
- If you worked alone, no need for the Collaboration.pdf.

Part #1: UML diagram
Submit to the assignment link on BB.

Part #2: full program
Zip the Maven project (and PDF if you worked in a group) and name it with your netid + Project2: for example, I would have a submission called mhalle5Project2.zip, and submit it to the link on Blackboard course website.

CS 342 Project#2 Fall 2021

Assignment Details:

Late work is accepted. You may submit your code up to 24 hours late for a 10% penalty. Anything later than 24 hours will not be graded and result in a zero.

We will test all projects on the command line using Maven 3.6.3. You may develop in any IDE you chose but make sure your project can be run on the command line using Maven commands. Any project that does not run will result in a zero. If you are unsure about using Maven, come see your TA or Professor.

Unless stated otherwise, all work submitted for grading *must* be done individually. While we encourage you to talk to your peers and learn from them, this interaction must be superficial with regards to all work submitted for grading. This means you *cannot* work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own. The University's policy is available here:

https://dos.uic.edu/conductforstudents.shtml.

In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing

your program to another student, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. It is also considered academic dishonesty if you click someone else's iClicker with the intent of answering for that student, whether for a quiz, exam, or class participation. Academic dishonesty is unacceptable, and penalties range from a letter grade drop to expulsion from the university; cases are handled via the official student conduct process described at https://dos.uic.edu/conductforstudents.shtml.