Connect 4

Jonathan Rach

CS225-01

Final Project

4/22/15

**Problem Discussion:** The general purpose of the project is to test learned knowledge of File I/O, Exception Handling, Inheritance, and GUIs by designing a moderately complex program. I chose to create a game of Connect 4, as Connect 4 provides opportunities for all of these higher level requirements to be met. Connect 4 also allows for additional program specific requirements to be defined and met. By choosing to design Connect 4, I hope to meet all of the upper level requirements as well as to provide a firm challenge for myself as a programmer.

**Requirements:**

Upper Level Requirements:

a. The project will utilize a GUI to allow for user input and to display results. All user interactions must occur via GUI widgets and/or GUI menus.

b. The project will utilize file I/O.

c. The project will utilize at least one case of exception handling to recover from a condition that would otherwise stop program execution.

d. The project shall include multiple objects and include at least one case of inheritance.

Specific Requirements to my program:

1. The program will be a two player game that alternates players after each mouse click that places a checker.

2. The program will provide the user with the option to save the winner of the game to a text file. It will accomplish this in a unique window that will write to a file without crashing the program.

3. The program will allow the user to display the previous winner of the game from a stored text file. It will accomplish this by reading from a text file and displaying the previous winner in a unique window.

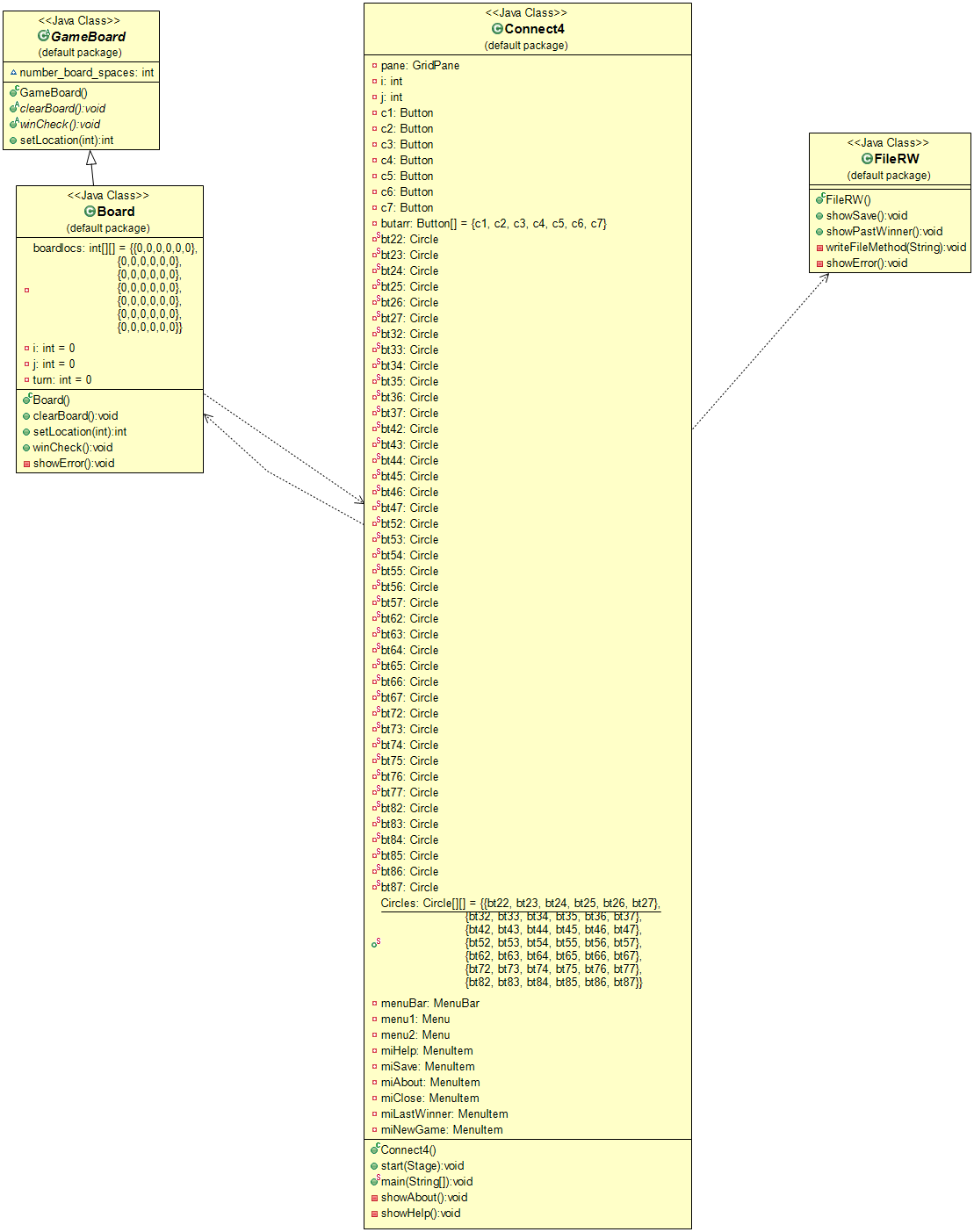
4. The program will allow placement of checkers only on the game board. The program will display an error message if the user tries to place a checker in a spot that the board will not allow, namely if the user tries to place a checker on the top of a full column.

5. The program will not place checkers directly over top of each other, that is, the program will allow only one checker per game space.

6. The program will display an “about” window that will contain information about the program. The program will also display a window entitled “Help” that will tell the user how to use the program and play the game.

7. The program will add checkers from the bottom of the board up, much as the real Connect 4 game starts from the bottom and works up.

**Class Diagrams:**



**Solution Discussion:** To solve this problem, I will create four classes. These will include one abstract GameBoard class, which will contain abstract methods to be implemented by all types of board games, one Board class, which will extend the GameBoard class and implement its abstract methods, one Connect4 class, which will serve as the main class, and one FileRW class, which will contain methods of writing to and reading from files. The Connect4 class will contain the main method that will create one instance of the board and one instance of the FileRW object and will create the GUI with all of its menus, buttons, and listeners. The board object will contain the main method for placement of each checker, which will follow the following algorithm: while (location[column choice][row] is not empty){

Increase the row}

Once an empty row of the selected column has been found, mark that array

location as occupied to place the checker there and change the GUI array so

that the GUI reflects a checker of the proper color being placed at that location.

This algorithm requires a try-throw-catch so as to prevent an array out of bounds exception. Once an array out of bounds exception is thrown, signaling the column is already full and a checker cannot be placed there, an error message is displayed and the array indices are reset so the program can continue. In addition, the program will decrement the turn counter so that a player trying to place a checker where it is illegal does not use up their turn.

Because I am not requiring the game to determine if there are four checkers in a winning combination and am leaving that up to user, I have made it so the user can save the name of the winner of the current game, which can be done through a menu item. This menu item triggers a new window which will save the user-inputted name to a text file. In addition, the user is able to view the last winner of the game, which is also accessible through a menu item that, when clicked, opens a small new window that contains the name of the last winner.

**Test Cases and Results:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Requirement | Scenario | Expected Result | Actual Result |
| 1 | A, D | The program is run. | The program will run and display the GUI. | The program performs as expected. |
| 2 | A, 1, 5, 7 | The program is run and two checkers are placed using the same button. | The program will place two checkers of opposite colors, one in the bottom row and one in the row above it. | The program performs as expected. |
| 3 | A, 1, 7 | The program is run and two checkers will be placed in random locations. | The program will place exactly two checkers of opposite colors. They will be placed from the bottom of the board up. | The program performs as expected. |
| 4 | A, C, 4, 5 | The program and checkers will be added until one column is full. Once the column is full, more checkers will attempt to be added to the column. | The program will run and print out a single statement alerting the user to the completion of a race. | The program performs as expected. |
| 5 | A, B, 2 | The program is run and the Save winner menu item is enacted. A winner name will be inputted. | The program will save the inputted name to a text file called “winner.txt.” If no file by that name exists, the program will create one. If one does exist, it will be overwritten. The program will not crash. | The program performs as expected. |
| 6 | A, B, 3 | The program is run and the display previous winner menu item is enacted. | The program will open a new window that will read the “winner.txt” file and then display the name of the previous winner as scanned from within the “winner.txt” file. | The program performs as expected. |
| 7 | A, 6 | The program is run and the “Help” menu item is accessed. | The program will create a new window titled “Help” that displays information on how to play the game. | The program performs as expected. |
| 8 | A, 6 | The program is run and the “About” menu item is accessed. | The program will create a new window titled “About” that displays information about the program | The program performs as expected. |

**Estimate Time and Lines of Code:**

I estimate that the documentation will take me a total of two hours, the coding will take me approximately 12 total hours, and the debugging will take me around 20 hours. I estimate that I will have around 300 lines of code for the program to run.

**Actual Time and Lines of Code:**

The documentation took approximately two hours of work and editing. The coding took me around 40 hours. Debugging time took around a half hour, but I built the program method by method and tested each piece as I added it, so much more debugging time was included in the coding time. Actual number of lines of code were 533.