**CIS 163 Project 2**

**A Connect Four++ program**

**Due Date**

* At the beginning of the lab; see the last page of the syllabus (i.e., schedule)

**Before Starting the Project**

* Review Chapters 6,7 of the CIS163 book
* Read this entire project description before starting

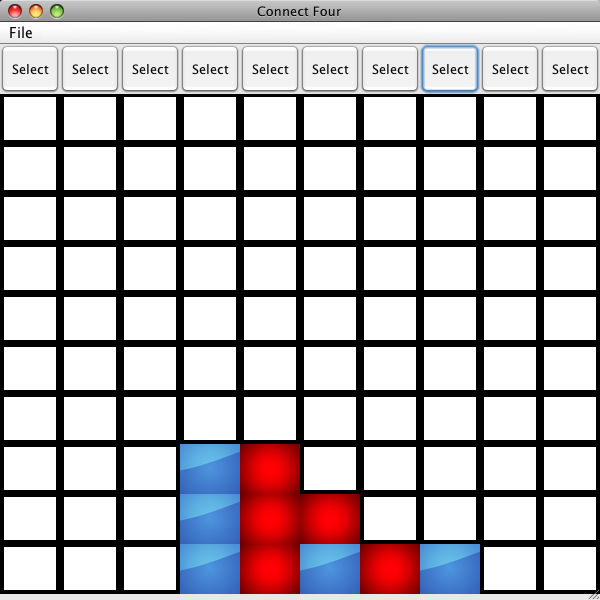
**Assignment Objects:**

* + To create a project that uses Model View Controller design (MVC)
  + To use a 2-Dimensional array of GUI components
  + To use a enum type within a programming project
  + To use nested loops to solve complex problems that involve 2-Dim arrays
  + To pass and/or return an array of objects

**Problem Statement:**

Write a Java program that plays the Connect-Four game and a bit more. **This basic game of Connect-Four will be demonstrated in class.** Remember, a player can win the game by having a horizontal, vertical, or diagonal connection. Listed below are the requirements for your program.

**Sample screen and this figure doesn’t include all the requirements found in this document.**



**You must complete each step fully before proceeding on. No credit is given to any given step unless the previous steps have been completed and is functioning!**

**Before you turn in your work: use the** [**Java Style Guide**](http://www.cis.gvsu.edu/studentsupport/javaguide) **to document your project. (10 pts)**

**Step 1: Create an Eclipse project named “ConnectFour”**

* Create a package named: package1 (right click on “Projects” and select package)
* Create a class named: ConnectFour (right click on “package1” and select class)
  + Chapter 6 has several examples of creating a main method and associated panel class.
* Create a class named: ConnectFourPanel (right click on “package1” and select class)
  + See Chapter 6 of your book for a typical example of a panel class.
* Create a class named: ConnectFourGame (right click on “package1” and select class)
  + This class contains all the methods for the game of ConnectFour and is shown in Step 6.
* Create an enum named: GameStatus (right click on “package1” and select enum)
  + See Step 2 for details.

**NO OTHER classes can be created without the instructor’s approval.**

**Step 2: Implement the enum class “GameStatus” using the following:**

There are only 4 possible states the game can be in: {Player1WON, Player2WON, Cats, InProgress}

**Step 3: Implement the class named: ConnectFour:**

Using the main program found in chapter 6 of your book as a guide, and create a main method that creates a JPanel object (**ConnectFourPanel)**. Create two JmenuItems: “*quitItem*” and “*newGameItem*”. Pass these two JMenuItem objects to the ConnectFourPanel. The “*quitItem*” JMenuItem quits the game, and the “*newGameItem*” starts a completely new game. (Note: For starting a new game, you cannot just re-instantiate a new game object, unless you dispose of the old Panel Object, instead, you are to re-initialize all the game variables and associated panel objects, see the instructor for more details if needed).

**Step 4: Implement the class named: ConnectFourPanel: Step 4 does the simple version of connect four with only a 10 X 10 board. Step 7 allows the user to enter in a different size board and different connection lengths.**

* Create the following properties for the ConnectFourPanel JPanel (more if you wish):

*private JLabel[][] board;*

*private JButton[] selection;*

*private ConnectFourGame game*;

*private final static in BDSIZE = 10*; // just a 10 x 10 board to start

*private JMenuItem newGameItem;*

*private JMenuItem quitItem;*

The JLabel variable “board” is a 2-dim array that represents the GUI board that the user sees. The JButton variable “selection” is used to select the specific column. *This.newGameItem* and *this.quitItem* are used to save off the two parameters, *newGameItem* and *quitItem* for later use in the actionPerform method.

* In the constructor for ConnectFourPanel(JMenuItem quitItem, JMenuItem newGameItem) do the following:
  + Create JPanels as needed so that you have a nice looking GUI display
  + Create JPanels as needed (e.g., panel), use a GridLayout that is 10 X 10 that represents the board.
    - You will need to use a nested loop, for example:

*for (int row = 0; row < BDSIZE; row++)*

*for (int col = 0; col < BDSIZE; col++) {*

*board[row][col] = new JLabel (" ");*

*panel.add(board[row][col]);*

*}*

* + Create listeners for every JButton in the 1-dim array variable named selection.
  + Use selection array to allow the user to select the desired column

*selection = new JButton[BDSIZE];*

*for (int col = 0; col < BDSIZE; col++) {*

*selection[col] = new JButton ("Select");*

*selection[col].addActionListener(listener);*

*bottom.add(selection[col]);*

*}*

* + Set the title of the JFrame title to “ ConnectFour”
  + Create a ConnectFourGameConnectFour Game object:

*game = new ConnectFourGame();*

* Create a private inner class named “ButtonListener” that implements an ActionListener. Create an actionPerformed method that calls the different methods in the ConnectFourGame class using the game object. For example, call the game.selectCol method when a user clicks a JButton on the board. The following is some of the code within the actionPerformed method that may be useful you:

// Determine which button was selected.

*for (int col = 0; col < BDSIZE; col++)*

*if (jButtonsBoard[col] == e.getSource())*

*// tell the game which button was selected.*

*// Game returns what row the chip falls to.*

*// Returning a -1 could mean row is full.*

*int row = game.selectCol(col);*

// Display information on the board using a simple setText method (Icon also could be used)

*board[row][col].setForeground(cList.get(*

*game.getCurrentPlayer()));*

// Determine if there is a winner by asking the game object. (see step 6)

*if (game.getGameStatus() == GameStatus.Player1WON) {*

*JOptionPane.showMessageDialog(null, "Player 1 won” +*

*“\n The game will reset");*

*}*

**Step 6: Implement the class named: ConnectFourGame: Step 6 continues step 5 and does the simple version of Connect Four game with only a 10 x 10 board. Step 7 allows the user to enter in a different size board and connection length.**

This class handles ALL of the game activities, and the following methods and properties must be created

* Create the following properties for the ConnectFourGame class (create more properties if needed):

*private int[][] board;*

*private GameStatus status;*

*private final static in BDSIZE = 10*; // just a 10 x 10 board to start

* public ConnectFourGame() A constructor method that initializes the board.
  + For example:

*status = GameStatus.IN\_PROGRESS;*

*board = new int[BDSIZE][BDSIZE];*

* public selectCol (int col) this method is called from the ConnectFourPanel class and is invoked when the user has selected a JButton. This method determines what row the chip will fall into.
* public reset() this method is called from the ConnectFourPanel class and it resets the board to a new game.
* public GameStatus getGameStatus() this method is called from the ConnectFourPanel class and it determines if a player has won the game after the select method (see above) was called. (The following are standard ConnectFour rules):
  + Return a GameStatus. Player1WON if player “1” has four in a row (Horz. Vert. Dia)
  + Return a GameStatus. Player2WON if player “2” has four in a row (Horz. Vert. Dia)
  + Return a GameStatus.Cats if all the Cells in the board are not empty and there is no winner
  + Return a GameStatus.InProgress if the previous rules do not apply

**Step 7: Add the following functionality to the game**

* Ask the user to enter the size of the board to be used and the number of connections to win the game (default 4). To accomplish this, you can use JOptionPane.showInputDialog method. For example, to ask the user to enter in ‘board size’ the following could be used:

String strBdSize = JOptionPane.showInputDialog (null, "Enter in the size of the board: “);

int bSize = Integer.parseInt(strBdSize);

the board must be greater than 3 and less than 20. (Recommendation: ask the user within the ConnectFourPanel class constructor). If the user enters invalid input (e.g., “abc”, -10) then warn the user with a message Dialog box and use a default values. Note: if the user selects “New Game” from the JMenuItem, have the game re-ask these questions.

* Add on to the main GUI display new JLabels that represent the number of times player1 wins and the number of times player 2 wins. (Initialize to zero when the games starts).
* Ask the user who starts the game first? Player 1 or Player 2. To accomplish this, you can use JOptionPane.showInputDialog (null, "Who starts first? 1 or 2“). Finally, if the user enters invalid input (e.g., “Pizza”, -1 then warn the user with a message Dialog box and use a default values.

**Step 8: Add on an undo feature:**

* Add on to the main GUI display one new JButton named: **undoButton**. This JButton, when clicked will undo the previous operation. Note, multi-undos is required, that is, continued clicking of this JButton will undo the game back to the beginning. Be careful to make sure the correct user’s turn stays in-sync after each click of the JButton.

**Step 9: Add the following functionality to the game. Difficult! (Partial credit given here.)**

**Previous functionality must still work, such as, win count per player, undo button and so on.**

* Ask the user to enter number of players (between 2 and 10, default 2). To accomplish this, you will need to remove the enum type GameStatus, and replace it with a method that returns the player that won. For example:

String x = JOptionPane.showInputDialog (null, "Enter in the number of Players: “);

* Add a different color and name for each player. It is suggested that you create an ArrayList of Strings for the different names of the players and an arrayList of Colors for the different player’s colors. For example:

cList.add(JColorChooser.showDialog(this, "Pick a color for player " + I, Color.RED);

--------------------------- YOU’RE DONE ☺ -------------------------------

**General concepts about MVC Design Pattern and these concepts will be discussed in class.**

1. In MVC design, the model, view, and controller classes work together to provide the functionality in the program. When a user interacts with the view (GUI), the following actions occur:
2. The view recognizes (via event handlers) that a user action (such as a button click, menu selection, etc.) has occurred.
3. The view then calls appropriate method(s) in the controller.
4. The controller accesses the model and calls appropriate method(s) on the model.
5. As a result of actions on the model, if there are any changes in the model, the model notifies the view of the changes. The view may also pull changes form the model as needed.

**Some additional grading criteria**

There is a 70% penalty on programming projects if your solution does not compile.

* Stapled cover page with your name and signed pledge. (-5 pts if missing)

**Late Policy**

Projects are due at the START of the class period, important, the first 24 hours late is -15 pts

* Each subsequent weekday is an additional -10 pts

**Turn In**

A professional document is stapled with an attractive cover page.

* Cover page - Your project must have a cover page that includes your name, a title, an interesting graphic or photograph related to the project topic and the following signed pledge: "I pledge that this work is entirely mine, and mine alone (except for any code provided by my instructor). " You are responsible for understanding and adhering to the [School of CIS Guidelines for Academic Honesty](http://www.cis.gvsu.edu/Academics/Honesty/).

**Project 2: Connect Four++ program**

|  |  |
| --- | --- |
| Student Name |  |
| Date Submitted, Days Late, Late Penalty |  |

|  |  |  |
| --- | --- | --- |
| **Graded Item** | **Points** | **Points Secured and Comments** |
| Javadoc Comments and Coding Style/Technique  (<http://www.cis.gvsu.edu/studentsupport/javaguide>)  Here are some examples but not limited to:   * Code Indentation (auto format source code in IDE) * Naming Conventions (see Java style guide) * Proper access modifiers for fields and methods * Use of helper (private) methods * Using good variable names * Header/class comments * Every method uses @param and @return * Every method uses a /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* separator * Overall layout, readability, No text wrap * Using /\*\* … / for each Instance variable * Has many inner “inner” comments | 10 |  |
| ConnectFourPanel class | 25 |  |
| ConnectFourGame class | 30 |  |
| Step 7: Additional functionality   * The board size can change * Any player can start the game * Number of winning connections can change * Labels to display number of wins for each player | 5  5  5  5 |  |
| Step 8: Additional functionality   * Undo feature | 5 |  |
| Step 9: Additional functionality   * Multiusers and Naming/coloring users feature | 10 |  |
| **Total** | **100** |  |

**Additional Comments:**