Basic Requirement Short Report

• Give a brief, top-level description of the program's major components and how they fit together.

ConnectFour – sets the initial conditions of the game.

InputUtil – used for asking the user for an input and validating that it is an int.

Model – used for setting the initial values for the board and also keeping track of player moves throughout the duration of a game. Also used for validating a move.

TextView – used mainly for representing the current state of the board.

Controller – used as a main loop. It connects all the functions from other classes together and puts them in a loop.

• How did you represent the board and why did you choose that representation?

The board is represented as 0s initially, then the zero's get filled with player's number as the player make their moves. I chose this representation because I found it's pretty simple and clear.

• How does the program keep track of which player goes next and how are players represented?

Players are represented by numbers 1 and 2. After each move the player in a variable is changed, and whoever is in the variable, that player's turn it is next.

I chose this because I think it's a really straight forward and clear representation.

• How did you test and debug your application?

I mainly tested my program as I went, meaning that I implemented something, then made sure that it gave the required output, I also made sure to input some unexpected values to make sure the program is robust. I tried implementing illegal argument exception and printing out stuff such as values passed into the program, however, where there was a few times where it helped, most of the time I found it was tedious and not really confusing.

• Which problems did you encounter and how did you solve them?

My main problem was the design pattern that was chosen for this project, I was not familiar with it and it was confusing at the stars, but after reading the code and doing some research in similar projects with the design I figured out how the program should flow.

• Are there any remaining issues? If so, do you have any ideas about how they could be solved?

For the basic part, I believe I have gotten rid of all of the issues, however I couldn't really test the program as it would be played in real life (ie with real players), so some issues could still surface, but they are definitely not any obvious ones I could think of.

Intermediate Features Report

The intermediate features I have implemented are:

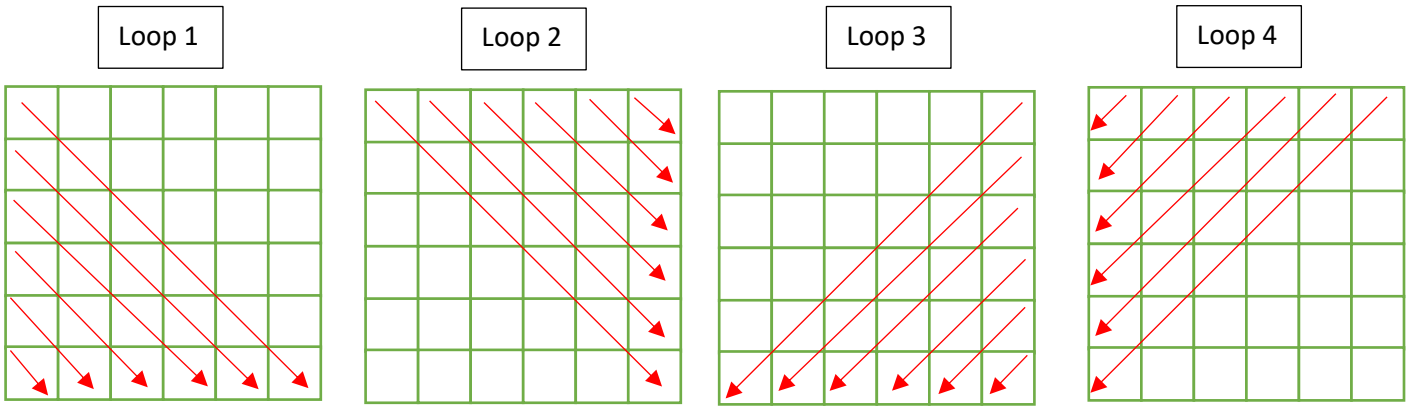
- **Allowing the user to start a new game.**
  I have put the main of Connect Four class into a separate method, which then gets called in main. This allowed me to also call this method in the controller class after the game is done based on user input. I did this because it made it really simple to initiate a new game.

- **Variable game settings.**
  Before the game starts, the user is prompted to input desired size of the board in terms of two variables: rows and columns and also a number in a row to win. Altering the parameters of the model constructor was really simple and quick to implement and did exactly what it was supposed to.

- **Enhanced input validation.**
  Number of rows and columns can not exceed 100 and has to be at least 1. I chose 1 and 100 as limits because 100 is already not really a realistic number, but I could still think of occasions where people might want to play it as an additional challenge (Connect Four on a 100 by 100 board for instance) and I chose 1 as I found the concept of "Connect One" really funny and I think other players might find it funny too. In addition, number in a row to win has to be smaller than one of the sides, so that the game is winnable in at least one way.

- **Automatic win detection.**
  Win detection is of course based on the number that the user inputs at the start of the game. The algorithm checks for the horizontal win first, then the vertical win, and then both the ascending and descending diagonal wins. The horizontal and vertical checks were pretty straight forward and are implemented with a counter that goes through rows and columns one by one.

  Diagonals were a bigger problem as I couldn't think of a way to implement them in one loop, so I created four loops that each go through separate part of the board in a different direction like so:



This isn't an ideal solution as depending on the wining number some of those arrows are redundant, so a possible upgrade would be to make sure that the algorithm only checks for winnable diagonals.

Advanced Features Report

I implemented the following advanced features:

- Save/Load the game
  Saving the game and loading it was done with java.io library and comma separated value text file. I chose this method as I found that it was simple to understand, not very long to implement and very much sufficient for what I needed it for.

  One problem that I didn't find a fix for is in the case when a player makes an incorrect move and then wants to enter the menu, because the loop that operates the menu would have to return two variables: move and breakorno, I wasn't able to put it into a separate method and ended up forcing a player to correct his move before entering the menu again.

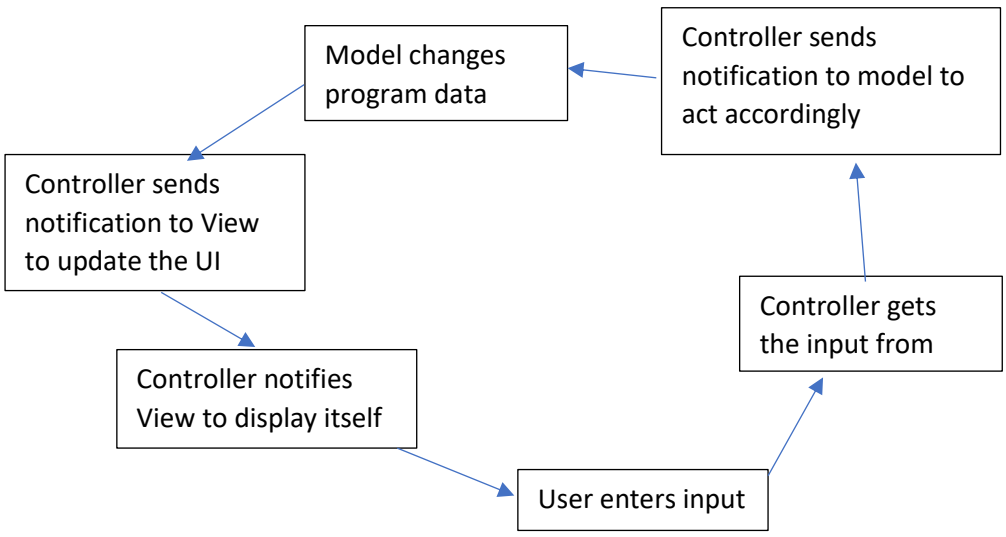
I have also researched the design pattern:

The MVC model has three main components: Model, View and Controller.

Model – stores majority of the data in the program and is referred to for manipulating and getting that data.

View – used to display data stored in the model to the user, it has access to the data but only for viewing purposes; it cannot change that data nor does it know how to. It is also able to acquire data from the user.

Controller – used for communication between model and view. Controller gets the data from view (data that user input into the program) and based on that data carries out tasks in model that modifies data stored in the program. Then once that data is changed, controller updates the view and then displays it back to the user.



The pattern is really good for making code reusable and more readable. In addition, because it separates the program into three different areas, it is easier to locate and solve any bugs. It is also more maintainable, as it would be more clear in the future which part of the program needs an upgrade.