Assignment 1 Report

s2058159

Basic Features

The Model class in my program contains the board, parameters related to the board (number of rows, columns and player) and functions for the different actions that can be performed on the board. These include the function makeMove which inserts a number corresponding to the player in the board.

The TextView class contains all input(taken using the and output functions. It performs all tasks that require direct interaction with the console. I added a function to display a message which is called when the preferred column is full and a function to display a message when the game is over.

The functions in Model and TextView classes do not interact directly and are called using the Controller class which consists of the functions startSession and gameLoop. The Controller class as the name suggests has control over the working of the game.

I chose to represent the board using a 2 dimensional character array as that seemed like the closest representation of a real time Connect 4 board. It could have been represented using an Integer array without making a major difference to the program. The board is displayed using StringBuilder, the empty spaces represented by ‘.’ and filled up spaces represented by a number (1 or 2).

I created a variable called player in Model to keep track of whose turn it is. Each player is represented by a number. 1 for the first player and 2 for the second player.
The gameLoop(in Controller) calls a function called nextPlayer(contained in Model) which switches to the next player at the end of every iteration. Each player is asked whether they want to continue playing after every move.

I tested the program by entering a number of different inputs and also by working out the run manually on paper.

Intermediate Features

I implemented all the intermediate features.
To allow users to start a new game I created a reset function in Model that would initialise the board to zero and set all other parameters to their default values. I also created a function called endSession in the Controller to prevent the gameLoop from getting too long. It calls the function to display game over message and asks the user whether they would like to play again. If yes, it calls reset and then startSession. If no, it exits the program.\

For variable dimensions, I created a setter function in Model. The user can choose to play on the default board or enter their own specifications. Input is taken using functions in textView. I created another variable connect in Model that depicts the number of discs to be connected to win the game.
There are some restrictions for variable dimensions to make sure the game can be played. The user specified variations are checked in the function isPlayable.

For validating the move, I checked whether it is within the limits of the board and if the column the user entered is already full.

Each move is checked for win in Model. I used a counter variable to check the number of connected discs and counters to depict the coordinates of the board. First they check horizontally, both in forward and backward direction. Then vertically, going down from the current move(as the current move is bound to be the topmost disc in the column we don't need to check downwards). I checked both diagonals in a manner similar to checking for win horizontally. I added a function in textView to declare the winner of the game.

I added some extra functionality concerning what would happen if a user enters the wrong choice. In this scenario, the program does not end, instead asks for another input.

Checking for win took me the longest. Initially I tried using for loops for the horizontal and Vertical checks which would check the entire row/column every time. But trying to find a way to check the entire board did not work out so I ended up using while loops for checking each direction from the current move.

Advanced Features

I managed a basic implementation of the “Play against Computer” feature.
It involved generating a random number within the limits of the board and automatically making a move.
The user is asked for their choice in the beginning. This choice is sent as a parameter to gameLoop which checks it against whose turn it is through if/else statements.

MVC Pattern

MVC stands for Model-View-Controller. It is a way to structure our program by dividing it into separate components. It is a popular way of organising code. It segregates the internal representation and logic of the application from the information accepted from and presented to the user. It is an efficient way to keep the code maintainable and easy to read.

The Model includes the main logic and data. It is completely independent of user input.

The Viewer is used to access data from the user and to represent data on screen however it does not contain anything pertaining to the logic of the program or how the data is being used and does not perform any tasks on the input data.

The Controller coordinates between the Model and Viewer. It receives anything the user inputs in view and then calls the appropriate functions in Model for execution of the program. It is responsible for responding to requests of the user by passing them on to the model.

There are several advantages of the MVC pattern, the most important ones being:

1) It makes the code easier to read and edit by grouping similar functions together.
2) It makes it possible for different developers to work on different parts of the code simultaneously. For example, the way data is represented and input in view can be changed without disrupting the logic or overall flow of the program i.e. without bothering about making changes to the model or the controller.

MVC architecture was first mentioned in the 1970’s and has since become a standard way of writing programs. It is used extensively across many different programming languages and in web applications. It allows us to give a new modularity to our code and makes for good design.