INF1B - Assignment1 Report

UUN: s2064253

A. Overview

In assignment 1, it requires to complete the game-Connect4. There are five classes and three of them needs to be complete and used in the order of MTC (Model-TextView-Controller) to build up the game. In the class-Model, the functions used for the game to run are created and initialized. In the class-TextView, it contains most of the display methods here as they need to be print out for players to be visualized. In the class-Controller, the methods can invoke from the previous two classes; they can combine and build up the game process here, more details given in the sections below.

B. Basic

a. Board

For the game to be play, create a grid(board) using a 2D-Array and display it as dots ('.'), that is easier for players to view during the game. To do so, firstly, a 'for' loop is used to initialize the number of rows for the grid. Secondly, create another 'for' loop inside to initialize the number of columns of the grid. The board must be visualized by the players, finally, create and initialize these 'for' loops in the class-Model and import it to the class-TextView for display.

b. Player

Connect4 is a two-player connection board game, then it needs to make sure both the players can place their chess alternatively within the range. A char-array with 'A' and 'B' will display as PlayerA and PlayerB used for alternate. Also, there are two methods called isColFull and isMoveValid used to strict the chess; it must be placed inside the range of the free columns in the board.

c. Main Game Flow

The class-Controller used to build up the main game flow. The Scanner can help read the input from Players. At first, for every chess the player is placed will run through isColFull and isMoveValid to ensure the chess allowed to place, otherwise, it will ask the player to place the chess again. The method-makeMove used to replace the '.' with corresponding 'A' or 'B'. For every chess placed, the game will display the grid with the newest update, this can help the player to control the game easier. Finally, Game flow needs to recursively execute with 'while' loop until it satisfies the end condition.

d. End Condition

There are three conditions for the game to end, one player wins or concede and the game to draw. Win detection will be in the intermediate section below. For concede, Boolean conditions set up for every fifth turn, the game will ask the next player if they want to give up; if a player said 'Y' or 'y', the game will end and the winner goes to the other player. For the game to drawn, the system needs to detect the board is full but nobody wins or concede.
Both the conditions are written as 'if-else' statement with Boolean functions used for conditions; as it works better than other loops. It will first detect if someone concedes and then test if the board is full for design needs. The 'break' in the 'while' loop used to end the game if either condition is satisfied.

e. Missing Features

A series of methods in the TextView used as guidance for the game. It also helps the class-Controller to be more clear and easier for view. Each method name is based on its function, so the editor can understand what is happening without a look back to the class-TextView.

f. Conclusion

1. How did you test and debug your application?

   To test and debug this application, modelling and testing different situation to make sure the game is working as the design pattern. Some of the real players are invite to play the game via social media, which helps to detect the real situations to see how the game will crash or not.

2. Which problems did you encounter and how did you solve them?

   There are some problems which take lots of time to solve. The majority problem is caused by 'while' loops. For example, input some character or integer that is not allowed or as require will cause the game crush. According to the design plan, this should output a statement to clarify this is not allowed and ask the player to input value again. However, the game is either end directly or jumps to the next player and crash down.

   By looking through the class-Controller, analyse the 'while' loops and 'if- else if- else' statement conditions to make sure they are in the right order so that they can perform the design functions.

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

There are some of the display messages as guidance for the game in the class-Model. As mentioned in the Overview, these should be inside the class-TextView. But according to the order MTC, the methods can't import from class-TextView to class-Model. Also, it can't build up in the class-Controller, as these needs to be print in the beginning before the grid initialized.

The problem may solve by move the board initialize to Controller and use a 'while' loop to link the display message for TextView and player input from Model.

C. Intermediate Feature

a. Start A New Game

The method resetGame has the design to use for initialize game state in the class-Model. For example, an empty board, initial player order and size of the board. Once the previous game is finish; the 'while' loop will run into a method called newGame in class-TextView that will ask the players if they would like to start a new game. If the answer is 'Y' or 'y', the game will be reset. Otherwise, the System will exist and print 'Thanks' statement. Use of this design is because the 'while' loops are more efficient and clearer for any editing in the later.

There are not many difficulties for this part of the game. But as the design plan; ignore the uppercase and lowercase difference is necessary, as it can give the players better experience. This can achieve by using toUpper method or equalsIgnoreCase method. The equalsIgnoreCase is better here as the Scanner is imported; so, it can use directly without any new methods.

b. Variable Game Setting

In the class-Model, three variables (nrRows, nrCols, ConnectX) used to represent number for rows, columns and winning connections, and to replace 6,7,4 (Connect4). This is also contained in the method restGame, for every new game, the player can choose the board size they want.

c. Input Validation

There are some board sizes can't be used for play. To eliminating the game crush, firstly, a Boolean method called validGridSize is created in the class-Model. If the board size wrong, then the method 'rule' in the class-TextView will display and tell the player how a valid board size should satisfy; and finally, the 'while' loop will ask the player to input again until the correct board size chosen.

Some rules for the board size have been found out, but also, some rules haven't figure out. As during the testing, the board size failed in 7 Rows, 6 Columns and 4 Connections.

d. Automatically Win Detection

There are four situations for the player to win in this game: horizontally, vertically, left diagonally and right diagonally. Using two 'for' loops for method 'hvWins' in class-Model as this method contains two situations: horizontal and vertical. And there are two different methods below to represent the other two situations. The grid is created from top to bottom and left to right; such as the top row has index 0 and also, the column index is increasing as it counts from left to right.

In the diagonal method, a third 'for' loop as 'Range' is introduced; this variable is used to detect the range of the diagonal line. Therefore, any diagonal lines inside the range and satisfy the number of connections can be detected to win the game. There are display winning messages inside the class-TextView, and the loop is built up inside the class-Controller. Three class work together to make sure if one player wins the game, firstly, it can be detected; secondly, the system will tell the player who wins and then ask if they want to start a new game.

D. Advance

a. Play against the Computer

The System will ask in the beginning whether the player wants to play with the computer or not. If not, the loop will be continuing as basic and intermediate features.

If yes, inside the class-Model, a method called 'getRandoms' which will generate random numbers in the range from one to the number of columns. This number will then need to check if it is placed in a free column if not, it will keep generating new number until is valid.

Some new methods for play with the computer create in the class-TextView and class-Model. For example: as the computer will not concede; add some new concede methods to ensure it will only ask the human player wants to give up or not.

Here are some ideas about how to improve this. Inside the class-Model; the 'hvWins' used a variable called 'connect' to count the number of connections between each type of chess. Firstly, if the AI could count this, determine the winning rate and place the chess in the column has the highest winning rate. Secondly, if it could count the

human player's connections and placing the chess in the columns to avoid the human player winning. Then, the AI will have a much higher chance of winning.

E. Evaluation

There are many difficulties as a Java beginner, such that the play against computer is quite a challenge. For this assignment, learning how to construct a large project as a whole and divide the different project into pieces is significant. MVC used to separate the methods, player display messages and the game flow. Knowing the order of MTC helps to understand the given code and also helps to be clearer. For any future editing, it helps the editor to pick up the pieces and better understanding.