Inf1B - Assignment 1
S2061204

Overview

The task for Inf1B - Assignment 1 is simple, create a connect 4 program in java with optional special features such as a bot to play against, saving progress, win detector etc. This project was created to test my knowledge in OOP from what I have learned so far in Inf1B. My project successfully achieves all the basic and intermediate features, which we will discuss shortly. My plan of action is to just attempt it straight on as I feel I am competent enough with Java to meet the criteria. I plan to create clean code that is readable and will be structuring my code into simple methods to improve its useability.

Basic Features

My program achieves all of the basic features, let's go through these features and how I created them.

1, 2: I started by familiarising myself with the code and files, so I had a concept of how the program would work.

3: I then had to do some research into connect 4. I have been creating chess AI's, so I overthought the input system to be an array of coordinates and then realised it was just the column input. I initially defined the userturn as an int (to represent each player), and if the game is over (win) as a boolean. I also defined an empty 2d array to act as the grid, as I thought this was the most straight forward option.

4: I chose to loop each row (starting from the top so it was displayed properly) and then inside that loop for each column so I could display every element. If that element was null it would display a space, and otherwise it would display the players corresponding token ie X,O.

5: Initially the make move function checks if the move is valid. It does this by checking if the board is full (by looping through each element and checking for null), if the column input is invalid (invalid or full column). If the move is legal then it will change the corresponding position in the 2d array. It then changes the player turn variable to signal the next player's turn.

6: I found this step relatively simple. I started by looping while the game is not finished, the terminal will then output who's turn it is to improve usability. It will then receive and input from the player and execute that move (checking its legality first). The program only then checks the board to see if someone has won and ends the program if so, but this is discussed in the next requirement.

7: This step was also relatively simple as I already had the check full board method. I added an output that told the user if they want to quit press 0, as 0 can never be one of the columns (the columns start at 1). This meant I had to change some functions to add a special clause that accepts 0. Once I had changed these methods, I made a make win method that changes the win variable to true if the conditions were met.

8: I tackled validation and errors as they came through 1-7 so there was not much I had to do here. I changed the row line for the print board so it looked nicer, but that was about it. Everything else was bug free.

Intermediate Features

I attempted all the intermediate features, and they were successful with one exception, but we will get to that.

Enhanced input validation: I found this relatively easy. All I did was create a restart method in the model that restarted the rows, columns, userturn, the win and the board to its initial values. Something I realised was I forgot to add the X in a row to win variable to this, this is a remaining issue. I decided to use a method as it looked cleaner and made the controller more simple and easy to read.

Variable game settings/ Enhanced Input Validation: This was easier than I thought. I originally thought I would have to change a lot, but I used the custom defined variables instead of constants like 6 and 7, so implementing this took nothing more than adding an input into the controller to ask the user (and validate) the number of rows and columns. I had to make sure that the connect X variable was less than the highest number out of rows and columns. I originally used pythagoras to try and solve this, but quickly realised it would not work. I solved this by creating temporary variables and only then assigning the temp variables to the model dimensions if they were valid. Apart from that my input validation already worked well so I did not have to change much. I used multiple extreme and exceptional test cases to test my validation.

Automatic win detection

This feature was my biggest problem, and ultimately I ran out of time and only partially completed it. The detection method works but only for connect 4, not connect X. I actually created this project a few years ago in python, so I went back to check the algorithm and translated it to Java. This algorithm only worked for connect 4 though, and I was unable to update it. The algorithm first checks all the horizontal and vertical lines, then both directions of diagonals (it achieves this by looping through every element, similar to other functions). I thought about removing this as it will not work if the user enters a connect X that is not 4, but I decided to keep it in as it shows I have attempted this, and it partially works.

Evaluation

Overall, I thought this project was a success and I managed to implement all the basic and intermediate requirements successfully with the exception of one. What I could do differently next time would be to definitely fix the win detection algorithm, as it is the only method that needs obvious improvement. I would also look to implement some of the more advanced features. I have already implemented these features into my chess AI so I feel like I could use that experience (of bitboard, minimax etc) to achieve these features. I found this project relatively easy, I was just limited for time and could not finish the optional requirements.