Software Design and Implimentation 1 Coursework Report

‘Connect 4’ project

Robert Shippey
N030543

Software Specification

The software that I will be creating is a C++ and Gwin version of the classic game Connect 4. The game is required to have two players to drop coloured coins into columns with the aim of making lines of four coins vertically, horizontally, or diagonally. The program should take the two users name and prompt them to take their move, repeating this for each players turn until a winner is found. Once a player is found the program will promt the users as to who won. It will then add the name of the player who won and the amount of turns it took for them to win into a score board. Once this is printed to the screen, it will save it the top ten scores to a file ready to be loaded when a new game is won.

Analysis

From analysing the specification, I have found four discrete elements that I can work on to sucessfully create the application.

-A game loop. For the game to function, most of the application will need to be in a loop to enable each user to take a ‘turn’ and to populate the board with each users coins.

-Adding coins to the screen. There will have to be processing to check, once a player chooses a column, where the new coin needs to be placed. If there are three coins already in that column the new coin will need to be placed on the fourth position.

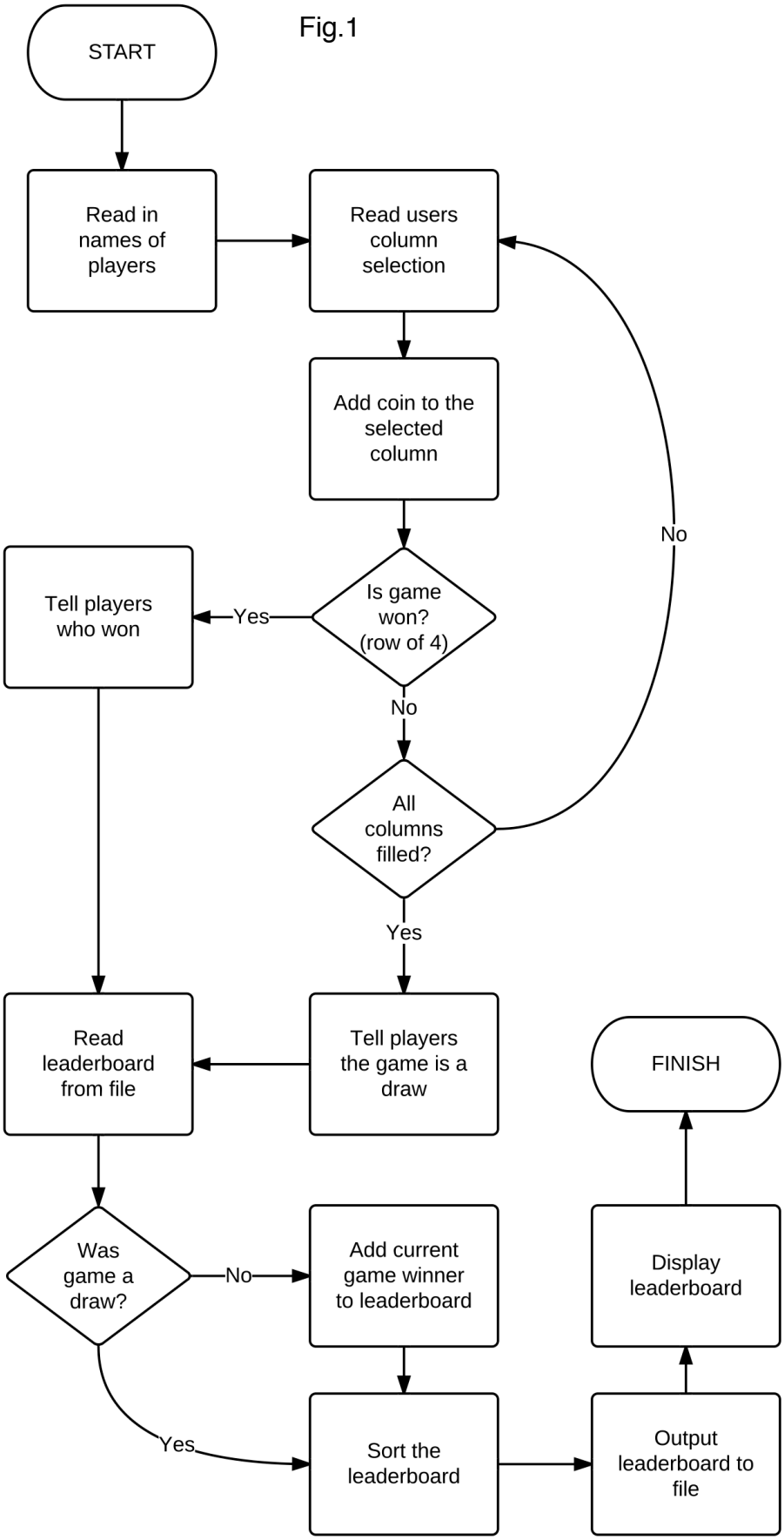
-Checking for rows of 4. After each players turn, the program will need to check if a row of four coins has been found and so someone has won.

-File handling. The game will need to read in the old scoreboard, and add the current current winner. Then, it will sort the scoreboard so the top winner is at the top then write the first 10 scores into a text file.

Design

To best demonstrate the design of the program in this project, I have created a flow diagram (fig.1) showing the processes and decisions involved.

Fig.1


START
Read in names of players
Read users column selection
Add coin to the selected column
Is game won? (row of 4)
Yes
Tell players who won
No
All columns filled?
No
Yes
Read leaderboard from file
Tell players the game is a draw
FINISH
Was game a draw?
No
Add current game winner to leaderboard
Display leaderboard
Yes
Sort the leaderboard
Output leaderboard to file

Initially, there are a few header files that I will need to include in the program to enable certain functions. The String header will be nesesary so that the program can read in the players names and easily manage those strings within the application. Also, the fstream header will be needed so that the leaderboard can be read in from and written out to a text file.

The main game loop is demonstrated on the flow diagram by the decision diagram with the question asking if all the columns are filled. This would be implimented in the code by a while loop that has the condition of a counter, starting a 0, less than 25.

To add the coins onto the game board the program will have to check the selected column for coins already in that column. A structure will be defined that will contain an array of 5 integers. In the program an array of 5 of that struct will allow a 1 or 2 (showing which player put the coin in) to be stored. When a player selects a column, the application will check if there is a 1 or 2 in the first place in the selected column it moves on to the next place, if there isn't it will add the players number to it. I will then use the Gwin circle function to draw the coin.

Once the player adds their coin to the board the application needs to check if there are a row of 4 anywhere. The application will check if rows, column, and diagonal connections of 4 places are equal to the number of the current player. If it is then the application should break from the main game loop.

After the game is played the application will update the leaderboard. An array of 11 structs that contain a string to hold name and an integer to hold the number of turns would be created. The previous leaderboard in a text file will be read and be held in the first 10 positions in the struct. The 11^{th} will contain the current players score. After the positions are all filled, the array will be sorted using the bubble sort algorithm. Then the first 10 are written back into the file.

Implimentation

Whilst coding this application, some of the elements were easier and some I struggled with. An example of an element that I struggled with would be testing for diagonal rows of 4 coins. I tried to cut down the amount of lines I had to write by constructing the tests in a loop, but the indexes for each column and position are difficult to calculate so I wrote them out by hand. Another area that I found difficult was reading in the leaderboard from a file and then writing the leaderboard back to the same file. I had to look at the problem very methoically and think about what needed to be done when so that I wasn't opening the file to write too early or reading when I intended to write. Any other issues that I came across were quickly rectified with the use of debugging techniques such as breakpoints.

Testing

Certain tests need to be carried out to ensure that the program works as described in the specification. Initially I did internal testing, ‘playing’ the game against myself to ensure that all events were triggered correctly and in the right order. This ensured that all the functionality was in place and that any minor mistakes that I may have let slip through in the coding would be identified and could be corrected before any users tried out the game.

Once I was confident that the software was bug free and suitable for user testing, I asked peers to play my game to ensure that the interface was easy enough to understand and that no new bugs would be found when the users tested it. One thing I learnt from the user testing was that the program needed a way of creating the scoreboard text file if it doesn’t exist because the program doesn’t work unless there is a file in the directory already. By distrebuting the release version of the game, I found it easier to only need to send the one exe file than the score.txt along with it.

Critique

I am personally very pleased with the software I created. I found the project interesting and challenging. I feel that the program is relitively large and complex, the code doesn’t have any unnecesary repetetive elements, and the Gwin output is clean and easy to understand. I think a few sections of the programs code could be neatened up, such as the mechanism for keeping track of which player just took their turn. It feels inelegant and clumsy and that could be improved.

To improve my software further, I could do a number of things. Firstly I could improve the graphics and animation in the program. An interesting addition to the animation would be animating the coins dropping into the columns. Another improvement I could make on my game would be to allow users to click the column to drop the coin rather than type a number to represent the column. It would be more intuative and would feel more realistic.

Declaration

I, Robert Shippey, do decree that all software submitted to the /cw reposity for this modules coursework is my own work. No code has been used that has been written by another party, except for any functions and classes from the Gwin library as it is a requirements that this library is used.