Niall Swan

40207307

ADS Report

Introduction

The task that had to completed was to make a Tic Tac Toe game that could be played by two players.

The features that my tic tac toe game includes is the ability to play against another player and play against the computer.

Design

When it came to designing the game, the first thing I implemented was the board. For this I started off by making an enum holding values of the 4 different board squares: noughts, crosses, border and empty. I then made an array of integers to hold the board locations that were playable. It then uses a board initialize method to create the board locations. The board is 25 positions in total, but it uses a border so only 9 positions out of the 25 are playable and the rest is the border of the board. The board is then printed with a print board method which contains characters to use for the board pieces and will print the squares of the board accordingly with what piece is on them (or if they are blank).

I then implemented the ability for players to place pieces on the board. For this I implemented a method that gets the players move based on their input and what squares are available. The user input will be checked to see if it is too long, not a number and if it is not between 1 and 9 before the value is returned. If it fails any of these checks the user will be prompted to insert a correct value. Once the value is returned a separate move making method will process it and place it on the board.

To make the players take turns I implemented a while loop in the method which runs the game. While the game has not ended it will see which sides turn it is and run their get move method, then the make move method. It will then change the turn to the opposite side which will do the same. The game will go back and forth until someone wins, or it results in a draw.

To detect when the game has been won I use a method which finds 3 of the same pieces in a row and returns an int based on the count of pieces found in a row. It will loop through each available direction and use a method to return any pieces in a row found in the current direction it is checking. If three pieces are found to be in a row the loop will stop, and it will return that 3 pieces in a row have been found. This will then end the game, display the victor and stop the while loop that allows players to have turns.

The game will result in a draw when a method, that detects if there are any empty spaces on the board, returns that there are no spaces left but the game has not been won by any pieces having 3 in a row.

As for implementing the AI I just created a method that will run instead of the second player and will randomly place a piece based on what spaces are available. It will loop through each space to find out what is available and then after it has all the available spaces it will randomly select one of them

to place its piece in. It will then return this, and the move will be made by the same method the player uses.

On the main screen of the game I gave the user an option to play vs another player or the computer by inputting 1 or 2. This will then decide which run method the game uses, the method for playing against another player or the method for playing against the computer.

Enhancements

If I had more time I would've liked to implement a more advanced AI that reads player moves and tries to counter the players moves and win as opposed to the random allocation AI I used. I would also have liked to implement a log of past games that could be replayed and a way to keep track of total player wins, possibly through a profile system that records wins of the individual players. Another feature would be boards of varying size/shape as that could lead to more interesting gameplay and add a bit of variety.

Critical Evaluation

A part that I feel works well is the way the program handles the player turns. The way that the get move and make move are in separate methods and not in the main method helps keep it neat and easy to understand. I feel that having it all contained within a while loop that keeps going, while the game is not finished, is efficient as once the game has completed it will stop running the code for the main game.

I feel that the board could be implemented in a better way, as to allow for different sizes to be generated easily without manually implementing each board. Alongside this the way of searching the board could be improved also, so that it automatically searches based on the current board size as opposed to fixed values for boards.

Personal Evaluation

I personally believe that this task has helped improve my confidence within C and my understanding of how to create console-based applications with C. I found it quite challenging to figure out which approach I should take for this task but I think the one I took can do what was required quite efficiently, but I believe there are many areas the program could've been made much more efficient and do the same with less code.

Overall, I feel I could've improved the program in complexity and functionality with better time management and better research into ways of approaching this task.

References

Bluefever Software – Youtube: https://www.youtube.com/watch?v=pt7XG2pvHa4&t=1s