**Tic Tac Toe Implementation in C**

Coursework Report

Algorithms & Data Structures (SET08122)

*Introduction*

The assignment was to write a game of tic tac toe in C, with a minimum requirement of implementing the following:

* Game board
* Players
* Pieces
* Positions

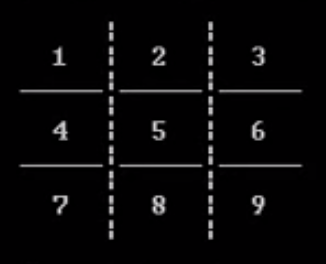
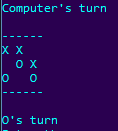
In addition to this minimum, I also implemented the following features:

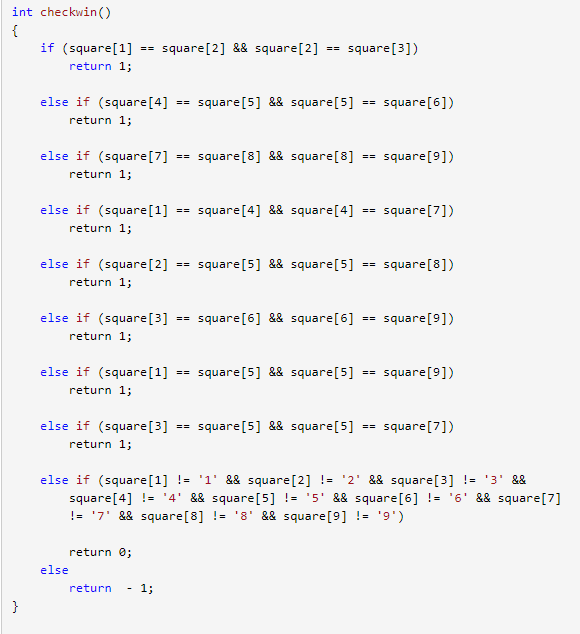
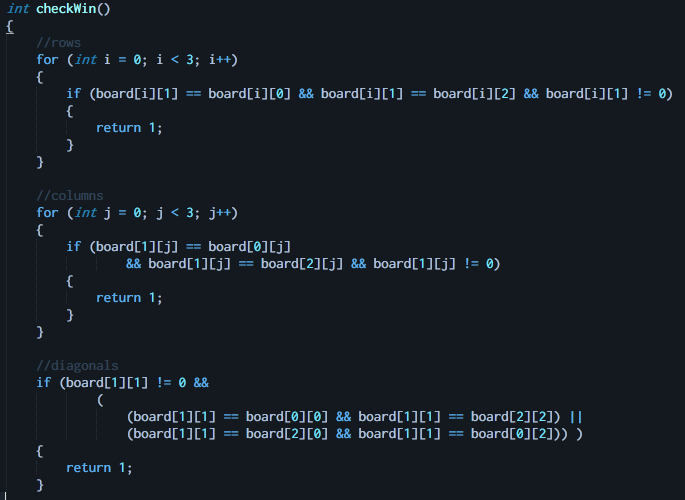
* Ability to pick which piece moves first
* Undo/redo of moves
  + Enable/disable of undo & redo feature
* Option of saving each game into a csv file
* Loading of previous game from csv
* Play against simple AI picking moves randomly from free spaces available

I partially implemented the replay of previous games from csv but the program crashes immediately after loading the moves into the game without an error message and I wasn’t able to fix it.

*Design*

My immediate instinct was to go with a 2D char array for the gameboard because it’s easier for at least myself, personally, to think of the board in a row and column manner similar to battleship or chess. After having a glance at some of the implementations online, I decided to stick with the 2D array as one of the only other viable choices, using a 1D array numbered 1-91 (pictured below) made for a less intuitive playing experience despite the increase in performance, which in the scope of what I was trying to accomplish would have been unnoticeable anyway. Another benefit of using a 2D array was the ease of evaluating win conditions, as well as making the saved game output easily understandable.

***Comparison of board appearance and win conditions 1D array gameboard vs 2D array***

I used the same variable for the players as well as the pieces, switching from ‘X’ to ‘O’ at the end of each turn, so I wouldn’t have to do any further work in setting the board pieces to the required chars. I couldn’t see any benefit in making it a bool which *would* smaller in theory but would stuffed to be a byte anyway, additionally it isn’t included as a native type.

I decided to create a “move” struct to store the game moves, comprised of the player that made the move (char) and the row and column number (int) to make moves easy to keep track of during undo/redos as well as being able to keep track of the moves in an array of structs and being able to easily reference a move by index.

I decided to keep the current player char, gameboard, and move arrays global because they’re accessed by so many functions it’d be a pain to keep passing them around, as well as the output path so it’s at the head of the file and easy to find.

I used a simple nested for to go over the board array & draw the gameboard, didn’t include any fancy extra lines to section off each space because I didn’t think it was a good use of my time and the gameboard is simple and small enough to not be confusing to navigate without them.

Both the one and two player modes use a for loop to play every round, the move being saved to the moves array and updating the gameboard. I decided the simplest and most intuitive way to accept user input for the piece placement was to simply read in two ints with a single scanf and check them for the correct range before assigning them to a move.

I made the undo & redo feature optional to save on time taken to play a normal game not requiring the feature. I implemented the undo feature by creating a second array of moves and copying the undone move at the same index as the current round before clearing the move and rewinding the turn order to the move before the undone turn. Similarly the redo feature was implemented by copying the move from the undone moves array back into the game moves array.

I didn’t implement any move validity checks in the game beside checking whether the desired space is occupied or not but would have added at least a legal move check to stop any losing moves being taken.

Each turn I checked for a win condition by looping over each column and row and comparing the middle cell to the adjacent ones (excluding 0 values). The diagonal check was similarly done by comparing the middle cell with the required adjacent cells.

The simplest way I could think to check for a draw was to call it if the win condition was not called by the time the for loop ended, as the maximum amount of moves possible is known to be 9, instead of checking over the board array for empty spaces.

I chose csv as the format for saving past games as it seemed the most legible and easy to save to and parse from. I managed to deserialise the csv back into a moves array for the purposes of replaying the game but ran into trouble as the program crashes here. Had I been able to resolve this, I would have been able to test and implement a replay mode with both automatic and press-key-to-continue playback; I left in the code for it regardless.

*Enhancements*

I would have liked to have built a smarter AI, including heuristics to ensure a draw each time, ie. putting priority on blocking enemy moves, disallowing moves that would ensure a loss next turn, etc.I would have also been interested in implementing at least a 4x4 board so the game would be actually fun to play, as a 3x3 is extremely easy to play to a draw unless one of the players is a child, or has not played before.It would have also been interesting to implement an AI vs AI mode to be able to evaluate the benefits of a particular strategy/heuristic by way of win percentages, as well as testing for performance of specific strategies. If I’d really run out of things to do I would have also added demarcation of cells on the gameboard as well as row and column numbering for easier reading of the board.

*Critical Evaluation*

I think it’s serviceable as a 2 player game of tic tac toe. The AI mode can be easily beaten as-is, however I did manage to lose to it unintentionally once during testing due to lack of attention. Saving the game moves to csv files works well and the produced output is easily understandable at a glance. Something went wrong, however, attempting to load a saved game back into the program for replays, as the program would hang and then crash without an error message after populating the moves array (I succeeded in iterating back over the populated array reading out the values of each field without an issue, but as soon as the function goes to return the whole thing crashes). I could have probably made a fancier gameboard and implemented custom game pieces but I don’t think these would have added much value to the game.

*Personal Evaluation*

I perhaps mis-estimated how much time to allocate to the coursework, had I given it more time I think I could have made it a lot more interesting and elegant.A particular challenge was my relative inexperience in C programming. This specifically gave me a lot of trouble when debugging and not being able to catch simple syntax errors as easily as I’m used to, the most obvious result of this being my lack of success in solving why the program is crashing on the loading of a previous game.Many of the bugs I lost a lot of time on were the result of a forgotten semicolon or similarly banal, and the ones that were not were usually solvable by a spot of googling, bar the mysterious errorless crash.Difficulties aside I feel markedly more familiar and comfortable working in C than I did prior to this project, and did derive enjoyment from figuring out how to go about writing the game, as well as satisfaction seeing the game in working order.

*References/Notes*

1 I’m referring specifically to this case, decided I definitely didn’t like this due to the massive amount of else ifs in both the win checks and move sets , beside the aforementioned benefits I saw in using a 2D array <http://www.cprogrammingnotes.com/question/tic-tac-toe-game.html>