Tic-Tac-Toe: A C Implementation

Lewis Wood

Edinburgh Napier University
40208881

1 Introduction

1.1 Project Aims

The aim of this project was to develop a text-based version of Tic-Tac-Toe in the C programming language. The project must be able to keep track of moves made by players as well as well as allow players to undo moves that they had deemed unwise after playing. As I did not want to allow players to keep undoing moves over and over, I allowed the other player to decide whether they should be allowed to undo their move to encourage more sporting play between opponents.

2 Design

2.1 Play board

I stored the play board in a simple 2D array of integers as this could potentially be expanded in the future if the Custom Setting enhancement (Section 3.3) was to be implemented in the future. I would have liked to have been able to use a Boolean array, however since there are three states that each square can be in, this was not possible. An array of integers also aids in win state detection which is detailed further in section 2.3. Blank squares were stored as a 0 in this array, Crosses stored as a 1, and Naughts stored as a 2

2.2 Move History

The move history was stored in 2 separate arrays, one for the X axis, and another for the Y axis of pieces placed. They are linked together with an integer value named “turns”, this allows the replay algorithm to recreate the moves made in sequence and works well considering my current experience with the C language. In hindsight this could have been stored as a linked list, or as an object, however my knowledge of C does not stretch that far at this time.

2.3 Input validation

To validate inputs, I compared string input with expected input using the strcmp() function found in the string.h file. I found this function to be quite straight forward to use, however it was not without its quirks (detailed in section 5.2).

When comparing integer values, it was simply a case of checking to see if it was within an expected range, and requesting the user to enter valid input if their value was found to be out of that range.

2.4 Win state detection

To detect win states, the possible square combinations that would ensure a win state had their integer values multiplied together. To detect a winning combination of cells, the program multiplies all winning combinations together, if the outcome of this calculation is either 1 or 8, a win condition is met, the game ends, and the winner is announced. To save CPU time, win conditions would only be checked after 5 moves had been made, as this is the lowest possible number of moves where a win state can be achieved.

2.5 Undo Function

The undo function was very easy to implement thanks to the move history storage outlined in section 2.2. The game simply resets the last modified cell in the array, reverts to the previous player's turn, and asks them again which cell they would like to choose.

3 Enhancements

3.1 Save Files

If I had more time to work on this project, I would have implemented a text file-based saving function, so that games could be saved and replayed by entering the name of the replay file in the command line of the console.

3.2 Artificial Intelligence

I also would have liked to have implemented an AI that someone could play against in a single player mode, this would have made an excellent addition to the project however in the time frame allocated, was not something that I could achieve.

3.3 Custom Settings

Thirdly, I would have liked to allow the player to designate the size of the board and the number of connected tiles required to win the game. This feature has regrettably not been implemented at this time, again due to time constraints, however it would be something I would like to see in the future.

4 Critical Evaluation

4.1 Features that work well

I feel the basic gameplay mechanics work extremely well. I had originally planned to use one big string to store the grid of cells for the game and manipulate it as required when a player made a move, however I am glad that this was not a feasible option as it would have been massively inefficient and a lot of code to write in C.

4.2 Features that work poorly

I feel the input validation could be improved upon, there have been some issues that required situation specific workarounds (Detailed in section 5.2).

5 Personal Evaluation

5.1 What has been learned

While working on this project I have broadened my knowledge of the C language as well as thinking of creative solutions to not having functions I am used to using in other languages. I found some of the issues faced interesting and learned a lot about some of the quirks of the C language.

5.2 Challenges Faced

Strcmp() irregularity.

While working on implementing the replay feature, I encountered an interesting irregularity when comparing entered strings from the user to “y” and “n” set values. I noted that even though these values were being entered, strcmp() would not equate them with one another. To overcome this, I was forced to compare the strcmp() result with 1 instead of the 0 used throughout the program to allow this user input to perform as expected. I’m sure that there are more elegant solutions out there, however

considering time constraints and my limited knowledge coding in the C programming language this workaround fits its purpose.

5.3 Personal Performance Review

I feel happy with my performance during this task, however I will admit that I struggled at times. These struggles were mainly due to a lack of functions that I would normally use in more familiar languages to do similar tasks, however these challenges allowed me to broaden my knowledge and deliver something that I feel meets most of the required criteria.