Console-Based Tic Tac Toe Game with AI Mode – Final Report

H00484499

Abbas Baig

1. Introduction

The individual software project developed for this project module entailed creating and developing a whole game in the C programming language. According to the given requirements for the game selection, the type of game that was to be developed was not specified and had the discretion of opting for either a text-based or graphical game. Based on such considerations, I decided to create a familiar game such as Tic Tac Toe and its console-based interface with an additional level of difficulty in the form of an AI opponent. The game was previously designed to be played with another person or the computer. The main goal of the project was to show the understanding of the basic programming constructs including the functions, conditional statements, loops, array, as well as the user input.

As for the idea of using the Borland Graphics Interface (BGI) for graphical output, it was purposefully decided to stick to the textual console environment. It was meant to focus on the game mechanics, gameplay, and the algorithm, in particular, its AI aspect that directly impacts the gameplay. In this report, the design justification, design process, the feature of the final game, and code organization of the Tic Tac Toe game will be explained with the decision-making process.

2. Design Rationale and Features

Tic Tac Toe was chosen because of its relatively simple rules; however, it has a rich inner logic when it comes to the AI aspect. The game is played on the board with each player using his turn to place his symbol the ‘X’ or the ‘O’ on the grid that is 3x3. The first player to get three symbols in a row, either in the vertical, horizontal or diagonal way, becomes the winner. If all the cells of the game board are occupied and no one of the players is the winner then the game is a draw. This simple structure of the game made it easier in terms of applying conditional checks, tracking the state of the board and taking turns by users.

To enhance the interest of the game as well as to make it more effective in teaching students, I had to add new features of one player game with the computer. This mode enabled the players to play against a deterministic AI, thus showing the capacity to implement game logic that entails decision-making by an algorithm. The AI functioned based on a set of rules in which the program searched for its win and then the prevention of the player’s win and then went to the next best move.

The last game included basic elements of the program, including the selection of the single player or multiplayer mode, the visualization of the board and the updated cell states, the error handling of input prompts, the win/draw detection and replay feature. This was done with an intention of making the program run continuously until the player opted to decide to exit – thereby replicating the actual game cycle.

3. Implementation Strategy

The game was implemented solely in C language and the compiler that was used for compiling the program was Microsoft Visual Studio. No other libraries were used, except the basic header's files, that is stdio.h, stdlib.h and stdbool.h to maintain portability and basic C programming standard. The code provided at the end of the program was divided into several functions that were well defined and organized and thus ensured modularity, reusability as well as easy debugging.

The game started by asking the user to choose his/her mode of play. As such, the gameplay loop was defined to either read the input from two players or compute the turn of the computer. The game board was implemented using a 2D character array of '1' to '9' which was used for the cells' identification. To make their move, each player was required to type a number from 1 to 9. On input, the program checked whether the cell where the piece was to be moved to was empty, and whether they were within a legal distance from each other. If valid, the move was made on the board, and the user was shown this on his interface.

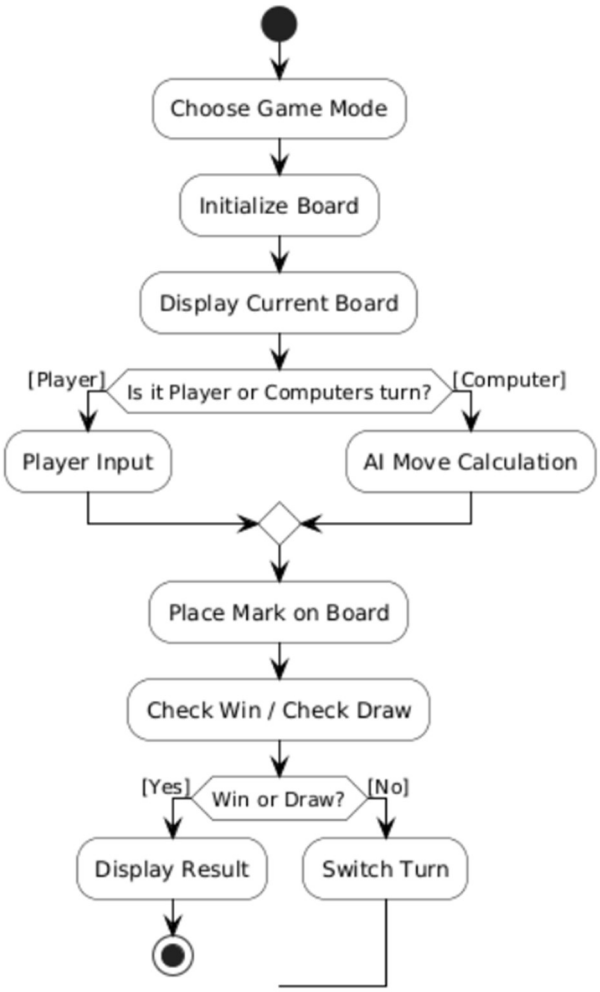
The AI move logic was to use a simple greedy algorithm. This algorithm first checked whether the computer could have a win in the current turn by making a check on all the possible moves. If there was no winning move, it proceeded and made a guess of the probable move of the opponent to check if a block is required or not. If none of the conditions were met, the AI proceeded to choose the next cell in the serial order. This made sure that the AI was able to play at its best at each round under the condition that it was not recursive and was following the Minimax algorithm.

After each move the program checks for any win or draw condition with checkWin() and checkDraw() functions. If a terminal condition was identified, it displayed the result and asked the player whether he or she wants to play again or quit. This process went on until the user decided to stop the process.

4. Technical Details and Flow Structure

Another feature of the implementation is the use of functions to compartmentalize various parts of the game. The initialization, rendering, decision-making, and validation processes were all allocated to their respective blocks. This gave the developer control over the game's logic and the code of the game was structured in an orderly manner.

The fact that it was a console-based game did not hinder the display of good feedback and instructions to the user. The use of system("cls") in the code erased the console before each move and gave it a more realistic graphical look to it. Also scanf_s() was used instead of scanf() to meet the requirements of the Visual Studio for secure code which minimizes the chance of buffer overflow and increases the input security.



The pattern of the game was rational and well thought out, which started with mode choice and the placing of the boards. It then continued in a cycle that switched between waiting for the input from the user or computing the AI move which depended on the current player. After every move, the board was changed, the terminal condition was checked and if the terminal condition was true, the result was shown, else the game continued. This flow was represented in a UML activity diagram that demonstrated the flow of control from one phase to the other in the game.

It's ensured that input validation was well checked to avoid overwriting of cells, dealt with the issue of improper characters and ensured that the buffer was cleared. Also, the replay logic helped to ensure that the memory was cleared for every new session so as not to compromise the board state of another match.

5. Reflection and Learning Outcomes

This project enabled the developer to put into practice all the theoretical knowledge learned in the C programming language. As with most games of Tic Tac Toe, the main goal is to connect three pieces horizontally, vertically, or diagonally before the other player does the same; this is where the complexity was in the programming of the application because one had to make it flexible enough to work with both human and artificial intelligence.

This was due to the introduction of AI, which brought in another level of overlay. Despite the fact that the implemented algorithm was not based on Minimax, it was possible to notice that it considered decision trees and conditional priority. Also, it provided a basis for further evolution in the direction of the unchallengeable AI strategy, which could increase educational potential of the game even more.

From an engineering point view, this project underscore the need of modularity, sanitization of inputs as well as code readability. This separation of logical functionality into functions proved to be useful since it eliminated code duplication and made the program more manageable and easier to test. Furthermore, to adapt the program for Visual Studio the developer got to know about some features of a particular compiler like scanf_s() and work with the runtime environments for different architectures (x86 and x64).

Thus, the decision to make the game text-based was justified by the absence of any distractions due to the interface and the overall gamification experience. Nonetheless, the project is highly

scalable. They added that in the following versions of the game, it could be programmed using BGI for graphical interface; mouse click detection; better AI using recursive algorithms.

6. Conclusion

The Tic Tac Toe game developed with console and an AI mode was able to meet all the goals set in this module. It was coded from scratch, concept to release, by the developer, so no engine, no borrowed code from other games. The project is an excellent example of how an individual can apply the knowledge of structured logic, memory management, user interaction, and integration of basic artificial intelligence into a C program.

The game was fun and engaging to play as a two-player game and also when one is playing against the computer. Due to its functionality, readability and safe coding standard compliance it was a good submission and easy to extend further. In summary, the project can be regarded as a good academic exercise in terms of learning programming theory alongside its application in game design.