Algorithms and Data Structures

Hari Kharel – 40312824

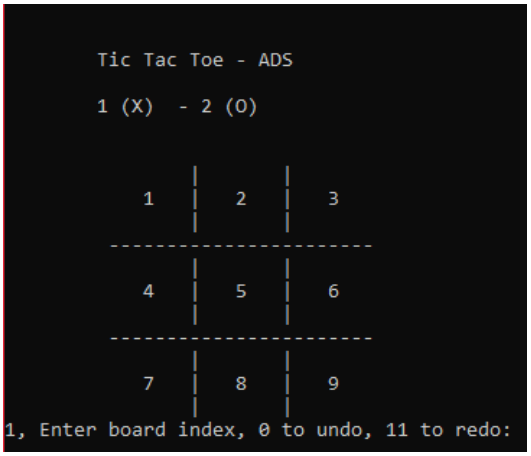
Tic Tac Toe Coursework

Contents

Implementation of Tic Tac Toe in C

Introduction

The base specifications defined some required feature for an implementation of a game called Tic Tac Toe in C. The game itself is a co-op game where either X or an O represents each player, most commonly played on a 3 by 3 grid. The goal of the game is to get 3 of your 'marks' in a straight line. The players move in alternating order.



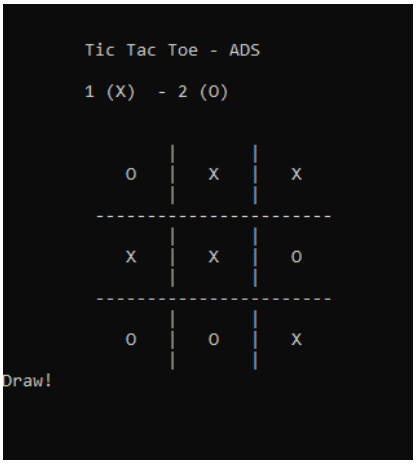
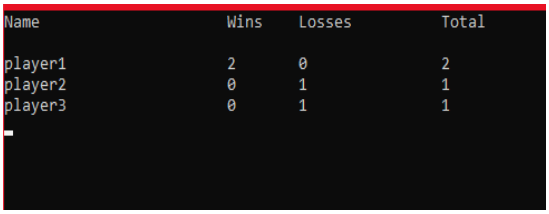


Fig 1.1, 1.2 Game Board of Tic Tac Toe

In addition to the required specifications, the implementation of the game also includes a host of additional features, such as the ability to undo and redo moves in a player vs player game mode, A scoreboard system that tracks all the wins and losses of a player, A replay feature that can replay any game between two players and finally, it includes a computer player.



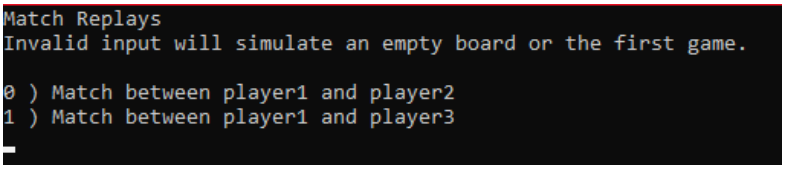


Fig 1.3, 1.4 Leaderboard Screen and Replay Screen

Design

Data Structures

The core design of the application is linear and revolves Array data structure, Loops and Structs. The implementation is written by keeping these in mind, these data structures were chosen as they are light weight and as simple to implement. This reduces the overall complexity of the code, and in turn increasing its readability as well as efficacy. Even though the computers these days are more than powerful enough to run this appellation easily, the simple design allows this program to run on micro machines, for example, Arduino.

Structs were used as the basis of the design is that a generic board is stored in a char array with the size of 10. Its only purpose is to show the user what's going on. The game moves struct is created every time a move is played, regardless of the player. It stores what board position the player moved their piece to, the players name and the players mark (X or O). When the game is concluded, the games moves struct is added to an array in a Match Struct. The said array holds 9 game moves structs, which is the maximum number of moves for a traditional tic tac toe game. The diagram below shows the different structs.

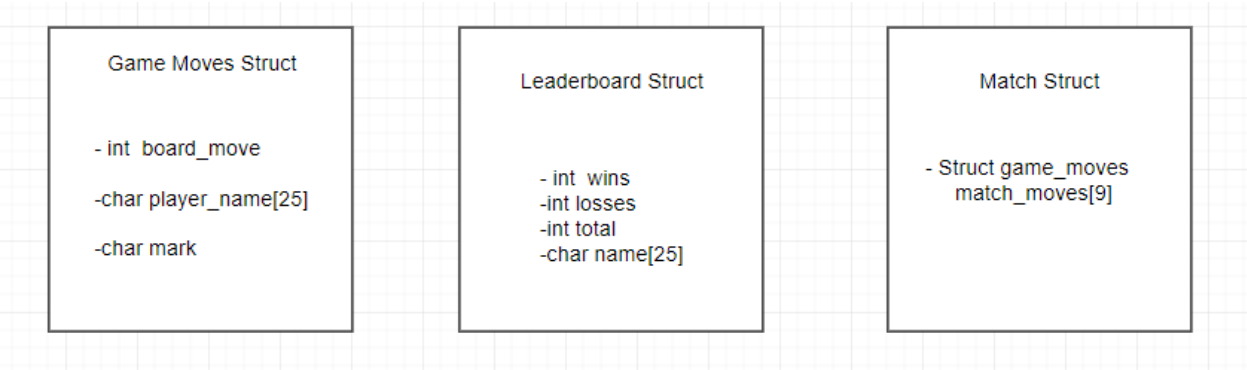


Fig 2.1 The different structs the program uses

Other storage types, such as hash maps, linked lists were available for me to use as well. However, with the rest of the application design, Structs were better integrated. The above mentioned Structs and arrays make of the data structure side of things for this application.

Algorithms

As for the algorithms that act on these data structures, they are modularized to carry out a singe task. This allows for plug and play kind of design where you are not restricted to one implementation method. This means that if you need to change how one bit works, you don't have to change half of your code to

achieve that. This allows for a better workflow and easier debugging as problems are usually only inside one method.

Player V Player Algorithm

The most prominent algorithm is the player versus player mode. It allows for two players to take turns and play. This is achieved in the pvp_mode method in the implementation. When called, it sets up some variables to start a new game. The main body of this method is a while loop that runs until the match is finished. The player moves, the win conditions, undo and redo are all checked there. When the required conditions are met, the program then executes the respective commands e.g. undo a move or next players turn.

```
199    void pvp_mode()
200    {
201        //setting up some vaiables, calling some methods to set up the game
202        fseek(stdin, 0, SEEK_END);
203        new_board();
204        get_player_names();
205        draw_board();
206
207        //function variables
208        int game_won = 0;
209        int player_indx = 1;
210        char mark = 'X';
211        char player[25];
212        struct game_moves moves[9];
213        int moves_index = 0;
214
215        strcpy(player, player1);
```

Fig 2.2 Implementation of the pvp game mode.

Player V Computer Algorithm

The basis of this mode is very similar to how player v player mode plays. The game alternates between the user and the computer. The user gets to pick an index as normally but, in the computers turn, another algorithm is called which returns a valid move for the computer.

```
while (gamewon == 0)
{
    fseek(stdin, 0, SEEK_END);
    system("cls");
    draw_board();
    int choice;
    printf("Enter board Index: ");

    //if its users turn, get the board index from them
    if (turn == 1)
    {
        scanf("%d", &choice);
    }
    //if computers move, call a method that returns a valid move for the computer to play
    else
    {
        choice = computer_move();
    }
```

Fig 2.3 The while loop of Player V Computer Mode

The implementation of the computer move algorithm takes a unique approach to how the computer player plays the game. The while loop is the main part of this method. Every time a loop runs, the computer generates a random number between 1 to 9. And each time a number is generated, the method then checks if the game board has that index free, if it does, the generated number is returned, if not, the process continues.

```
448     //method that returns a valid move for the computer
449     int computer_move()
450     {
451         int computer_choice;
452         int valid_move = 0;
453         while (valid_move == 0)
454         {
455             //generates a random number between 1-9 and checks if its valid or not until its valid
456             computer_choice = (rand() % 8) + 1;
457             if (board[computer_choice] == computer_choice + '0')
458             {
459                 return computer_choice;
460             }
461         }
462     }
463
```

Fig 2.4 Implementation of how computer picks a move

Print Leaderboard Algorithm

The leaderboard algorithm is a very simple one due to the usage of my chosen data structures. The algorithms loops through the Match Struct Array and simply prints the data stored in each of them.

```
516 void print_leaderboard()
517 {
518     system("cls");
519     printf("Name \t\t\t Wins \t Losses \t Total\n\n");
520     for (int i = 0; i < 100; i++) {
521         //checking to see of the leaderboard entry is populated
522         if (strcmp(leaderboard_array[i].name, "") == 1) {
523             printf("%s \t\t %d \t %d \t\t %d\n", leaderboard_array[i].name, leaderboard_array[i].wins, leaderboard_array[i].losses, leaderboard_array[i].total);
524         }
525     }
526     getch();
527 }
```

Fig 2.5 Print Leaderboard Algorithm

There are many other algorithms in the program such as the replay algorithm, move validity checker algorithm etc and due the limitation of the report length, they are not broken down here. However, the code is commented and formatted properly to make it easy to understand.

Enhancements

Due to time limitation, some of the features that were planned were not implemented in the final build. Some of those enhancements were; Persistence by writing to text files, AI player using Minimax algorithm to make it unbeatable and truly artificial intelligence. Another feature that could have been

implemented is a recursive game. The basis of this mode would be to play a maximum of 9 games and the board would be stored in a similar struct. As for playing this game mode. It could have been alternating, for example, if you placed a maker on position 6 of 1st board, the 6th board would be played next. The position dictates what board is played next.

Critical Evaluation

I believe all the features that were implemented were implemented well. Some of the features (such as serialization) that were being worked on were scrapped due to their improper function and due to lack of time to solve the issues. One of the issues with serialization of the use of nonstandard libraries for C, there were other ways around it but most of them were too complex to implement in a short amount of time.

Personal Evaluation

Overall, I feel like I performed well in this assignment as all the required features are implemented well. I have also implemented a host of additional features that were not needed to enhance the game. On top of that, the program offers a simple number-based navigation system to make the user interface as clean as possible. By making this game, I learned a lot about how C works, the strengths of C and the limitations of C.