Algorithms and Data Structures Coursework Report

Darwon Rashid
40280334@napier.ac.uk
Edinburgh Napier University - Algorithms and Data Structures (SET08122)

1 Introduction

Since the early days of computing, one of the most fundamental elements in computing is the science of data structures and algorithms. The reason why it is important is that it is relevant in almost every computing problem. There is always a need to represent some data or do something with it by a certain algorithm, so it is very important for anyone in the computer science field to have a good understanding of these fundamentals. The problem given for this coursework is to be able to create a Tic Tac Toe game with a focus on data structures and algorithms. The Tic Tac Toe game is just a medium for being able to represent data in a way that is memory efficient and speed efficient (given the context). The Tic Tac Toe game that was created for this course work allows the user to either play against another user or play against an AI (minimax algorithm). The user can even save their game and load it later to finish it. Users are given the option to be able to undo and redo their moves throughout the duration of the game.

2 Design

This coursework is focused on software design in terms of data structures and algorithms. This means that front-end(presentation of software) takes a back-seat and back-end takes the stage lights. The decisions made for representing certain elements in Tic Tac Toe have a dramatic effect on how much memory is used. Since Tic Tac Toe game is a relatively easy concept to implement, the challenge comes in choosing the data structures that work best given the context (in this case a Tic Tac Toe game).

2.1 Memory Management

Since this course work is done in C, a lot of the memory management decisions are given to the developer to make. Every variable that is allocated any memory by the developer, must have its memory be freed after use. the calloc method was used to allocate memory for when it allocates memory, it initializes it to zero, while malloc doesn't.

Listing 1: Initializing the list (2D array)

```c

char ** initList()
{
    char **list = calloc(1, MAX * sizeof(char*));

    for(int i = 0; i < MAX; i++)
    {
        list[i] = calloc(1, 2 * sizeof(char) + 1);
    }

    return list;
}

// freeing memory
void emptyList(char ***list, int rear)
{

    for(int i = 0; i < MAX; i++)
    {

        free((*list)[i]);
    }

    free((*list));
}
```

2.2 Game Board and Moves

The game board for a Tic Tac Toe game has nine positions to fill. There a lot of ways to represent this game board, an early approach was to have it as a 2D array and access each row and column individually (like this [row][column]). It was later changed to a single string of size 9 (in C, this is still an array, but as an array of characters). This approach saves space (uses less space than a 2D array) and makes sense given you know the size of the board. The advanced solution isn't always best solution, why use a solution that uses more memory but achieves the same results as a simpler solution. This is true in all aspects of the design for this coursework.

A move is a two character string that holds the given position and the move played at that given position. All moves when made after they are checked to be valid are appended to a 2D array that behaves like a list. This 2D array is the main data structure that holds all the moves made throughout the duration of the game. Initially, the data structure that was implemented to hold the moves was a doubly linked list. The doubly linked list was a Struct that had three Nodes (front, rear, and current) and a node is another Struct that held a string called move, and two more Nodes called previous and next (for undo and redo functionality). This proved to be more dynamic in functionality given the design of the data structure. However, since we already know much space we are using given Tic Tac Toe (nine moves), the benefits of its dynamic ability of memory allocation is thrown out the window for this context. A simple 2D array actually accomplishes the same thing and is faster and uses less memory.

The way the game keeps track of what player turn is by having a counter and then incrementing it by each turn. If the counter is odd, then it is player 1's turn (x), while if the counter is even, then it is player 2's turn (O).

2.3 Undo and Redo

For undo and redo, there was a question of how to represent them in the game, the 2D array (list) holds all the current moves played for that game session. One approach was to make lists hold the potential undo and redo moves, but since we already have a list that holds the moves played in order, there was no need to make new data structures to hold the undo and redo moves. There are three integers to indicate certain elements in the list, the integers are front, rear, and current. These integers are passed into the index of the list to return values that are relevant to what the integers mean. The front integer always returns the value at the front of the list (in this context, it is always the first move played), rear returns the value at the end of the list (in this context, it is always the last move played), and current returns the value of the current move (to keep track for undo and redo). What happens instead is that when a user undos a move, that move in the list is undone from the board and then the current move is updated to previous one and vice versa for redo (it is not removed from the list unless a new move is made after undo, in which case, it then removes the moves you can redo and adds the new move made as current).

2.4 Load and Save Game

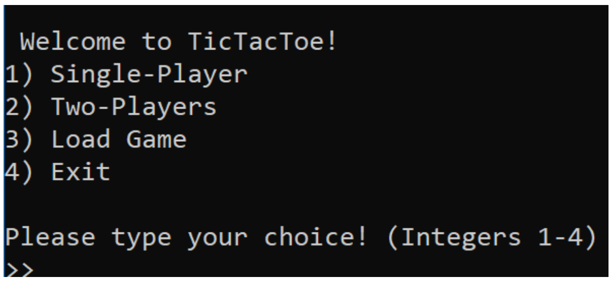
The game allows the user to be able to save a game state, and then later come back to that game state to finish the game. The user can even load in fully played games for replay purposes, once it is loaded, they can undo and redo the moves to look at what moves were played. The game saves the game mode (either single player or two players) and the moves played (until that game state) of a game into a txt file. The game recognizes what game mode it is when the txt file is loaded back into the game, and updates the 2D array (list) to hold the moves played for that game state.

2.5 A.I.

In the world of games, A.I. plays a very important role. In the early days, the use of A.I. has been to simulate a computer that plays with the user in a certain game context. Nowadays, A.I. is implemented within games for more complicated tasks. For this game, A.I. was implemented to play with the user. To decide on what algorithm to use for this game, there was a need to look into game theory for a place to start. After investigating game theory for this game, the minimax algorithm was used. The reason minimax was chosen was due to the fact that it is relatively simple and isn't computationally expensive for a game like Tic Tac Toe. There are not a lot of moves to consider, so minimax does not do a lot of computation. There was also no need to include pruning (usually bundled with this algorithm) for there are not many moves as mentioned above. Minimax will choose the move that is worst for the other player. The references include more detailed explanation of the Minimax algorithm.

2.6 Other Game Elements

When the program starts, they are prompted to a menu screen, they have four options, single player, two-players, load game, and exit. Once the user starts a game, they can type "help" to get instructions for the game, they can also type "save" or "exit" to exit the game and then the game will ask the user if they would like to save their game.



```
 Welcome to TicTacToe!
1) Single-Player
2) Two-Players
3) Load Game
4) Exit

Please type your choice! (Integers 1-4)
>>
```

Figure 1: Main Menu

3 Enhancement

For this coursework, we were given a task to implement a simple Tic Tac Toe game, there were other optional tasks that were given to us to implement, which I have done for this coursework. There are a couple features that were desired to be put in the game, but due to time, the extra features were cut out.

I wanted to add player names and information. Keeping track of how many games two specific players have played and the amount of wins, losses and draws between those two players.

I would remake the save game and load game feature so that it doesn't require the user to save it to a txt file, but have the game do it all automatically (like most games).

I would have also liked to add a feature where the user decides how big the board is (e.g. 4 by 4, 5 by 5) and how many of the same piece is needed to win the game (e.g. three X's or O's to win, four X's or O's to win). I want to change the checking if the game is won by checking if the game is won by every moved played, and not by checking the board for that takes more computation.

4 Critical Evaluation

When working on this course work initially, I had to think carefully about my approach, I used a doubly linked list, and then after evaluating it, I realized that it isn't better than a simpler solution like a 2D array. A linked list is good if you do not know the maximum size of your data structure, but given this game, we know from the beginning, how much space we will need, so a linked list does not save more space, in fact, it is more computationally expensive than a simpler approach.

I initially also saved every move as the whole board state which was a string of nine characters. This is unnecessary for you can save every move as a two character string, one character being the position in the board, and the other being the move played. This way, you save more memory, and the logic used for undo, redo, and other game aspects do not differentiate in terms of speed.

5 Personal Evaluation

Initially, I did the Tic Tac Toe game with a doubly linked list approach, and then realized that I was wrong about my approach, the advanced solution isn't always the best solution. This is when I understood the importance of understanding Algorithms and Data Structures and why it plays a big role in computing. Using C as the main programming language for this course work was challenging but very rewarding for it boosted my understanding given the low-level context of C. I learned a lot more about pointers and memory management, I can confidently say that due to this, it has made me a better developer. I now know to properly evaluate memory and speed thanks to this course work and module.

The labs given for this course work were very resourceful along with all the extra background reading given to us. Given this was a game, I also looked online on game theory and how board games are simulated in code. The references includes all the links to outside resources.

References

Minimax Algorithm references:

[1]
https://medium.freecodecamp.org/how-to-make-your-tic-tac-toe-game-unbeatable-by-using-the-minimax-algorithm-9d690bad4b37
[2]
https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

Tic Tac Toe information:

[3]
https://www.geeksforgeeks.org/implementation-of-tic-tac-toe-game/
[4]
https://inventwithpython.com/chapter10.html

Other information:

[5] calloc and malloc
https://www.geeksforgeeks.org/calloc-versus-malloc/
[6] strstr function
https://www.tutorialspoint.com/c_standard_library/c_function_strstr.htm
[7] doubly linked list
https://www.geeksforgeeks.org/doubly-linked-list/
[8] information on data structues
https://www.geeksforgeeks.org/top-algorithms-and-data-structures-for-competitive-programming/