

Coursework Report

Alexander Sims

40272671@napier.ac.uk

Edinburgh Napier University - Algorithms and Data Structures SET08122

1 Introduction

The task presented to me for the Algorithms and Data structures coursework was to create a Tic-Tac-Toe game which would allow users to play out a full game while also implementing other useful features including the ability for the user to be able to undo and redo turns they have made during the game and to allow games to be saved to an external file which can then be read into the program. When the game data is read into the program, the sequence of moves made is replayed to the user turn by turn. As this coursework is for algorithms and data structures we were required to pay close attention to the types we use for each of the key parts of a tic tac toe game. These are listed below

- 1.The game board in which the game is played.
- 2.The players of the game, i.e those interacting with the program.
- 3.The marks the user uses to make there turn (X or O).
- 4.Which positions the user has entered their mark onto.

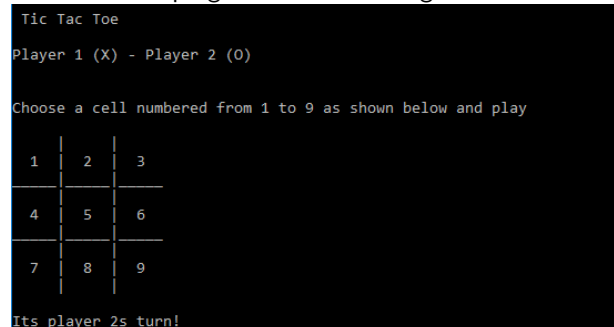
It was crucial that the game must be a feature complete Tic Tac Toe game that two users could sit down, run the executable and play without any issue. To achieve that end i used data structures which i knew would be suited to this kind of task. After having implemented the game using appropriate data structures and algorithms i believe i have met these requirements and produced a quality Tic Tac Toe experience.

2 Design

The first feature that i implemented into my program was the game board itself. I considered several options when deciding how to represent the game board in the program including a matrix, a 2D char array and a regular char array. In the end i decided to go with a simple but effective char array which held the values 1 to 9 as is the case in Tic Tac Toe. I chose to use this data structure over the others as it required the least amount of memory to store and work with and because it was the simplest to implement into my program. As this array is used to store the values of the board internally i needed to display something to the user which shows the contents of this array in a way that users are used to seeing in a Tic Tac Toe game. To do this i wrote out the game board using print statements which included the values stored in the board array. Below is the code i used to do this

```
1 void gameBoard()
2 {
3     printf("\n\n Tic Tac Toe \n\n");
4
5     printf("Player 1 (X) – Player 2 (O)\n\n\n");
6
7     printf("Choose a cell numbered from 1 to 9 as shown below ↔
8         and play \n\n");
9     printf("  %c | %c | %c \n", board[1], board[2], board[3]);
10
11     printf("----|----|----\n");
12     printf("  | | \n");
13
14     printf("  %c | %c | %c \n", board[4], board[5], board[6]);
15
16     printf("----|----|----\n");
17     printf("  | | \n");
18
19     printf("  %c | %c | %c \n", board[7], board[8], board[9]);
20
21     printf("  | | \n\n");
22 }
```

This created a game board which when viewed while the program is running looks like this:



```
Tic Tac Toe
Player 1 (X) - Player 2 (O)

Choose a cell numbered from 1 to 9 as shown below and play

 1 | 2 | 3
--|---|
 4 | 5 | 6
--|---|
 7 | 8 | 9

Its player 2s turn!
```

The second feature i added to the program was the ability to store the moves that have been made by each player. There were several possible data structures i could have used for this such as a simple queue or stack or a type of linked list. I chose to use a doubly linked list to store player moves as the ability to look at access the previous node was incredibly useful for implementing the undo and redo features which i will discuss later on in this report. The way it works is that after a player makes a move there move is appended to the linked list to be stored until the end of the game. When appending to the linked list i pay close attention to how much memory is allocated to each node and to the whole linked list. This along with freeing up each node in the linked list once a game is finished allows the linked list to use less memory than other data structures i could have implemented. Below is my code for freeing up memory in my list after a game has come to an end:

```
1 void freeListMem(struct node ** list)
2 {
3     struct node *currentNode = *list;
```

```

4 struct node *nextNode;
5
6 while ((currentNode) != NULL)
7 {
8     nextNode = currentNode ->next;
9     free (currentNode);
10    currentNode = nextNode;
11 }
12 *list = NULL;
13 }

```

This function allows me to dynamically free up nodes in the list no matter how many are used and keeps the amount of required memory low. This becomes particularly important if several games are played in a row. If i did not free up the memory in the linked list this could cause the program to become slower and could cause memory leaks. This function prevents this from occurring.

The last things i had to have in my game to make it a Tic Tac Toe game was players and their counter or mark (X or O). I discovered that i could represent each of these with the same value as the player will either be the X player or the O player and they will have the same mark depending on which they are. So to save on memory usage i decided to combine the two and just have a mark character either X or O which also indicates which players turn it is etc. Then to determine which players turn it was i just checked if the turn count was divisible by 2 if so then its player Xs turn if not its player Os turn. Alternatively i could of created a player struct and assign each player in the struct with a mark but i opted to do it the way i did for simplicity as-well as memory management.

I designed my program so that each individual part would have its own function. I did this for a number of reasons but the main one being to make the program as easy to read as possible and to reduce the amount of code to be ran from Main. Below is a list of all the functions i use to achieve my Tic Tac Toe game:

```

1 int checkwin(int board_Size);
2 void gameBoard();
3 void tictactoe(int board_Size, struct node **list);
4 void append(struct node **list, int num);
5 void freeListMem(struct node ** list);
6 int undo(struct node **list, int gameCounter);
7 int redo(struct node **list, int gameCounter);
8 void writeList(struct node **list);
9 void readList(int board_Size);

```

I have also made sure to include several error checks in my program to make sure each game runs as it should. An example of this would be my validation of the users input to make sure they have chosen a valid number and my check-Win function that checks at the end of each turn if a user has won or if all the spaces have been filled without a winner so the game must end in a draw.

3 Enhancements

There are several enhancements that could be added to my Tic Tac Toe game which would make it an overall better and more feature rich experience. The biggest improvement to my game would be the inclusion of a bot that a user could play against if they were without a second player. This

would not only provide another game mode for the player to engage with but it could also be used to make the game more complex as the bot could be taken a step further to include several difficulties for players to practice against. This would provide a challenge for the player and would increase the time the player spends with the game before running out of content. Following from that the next biggest enhancement i could add would be the addition of adding an online multiplayer game mode to the game to allow players to play against there friends over their network. This would be a major upgrade for the game as it would supply the player with a new more social experience playing against other real people rather than just the bot when they are by themselves. This would be quite a time consuming enhancement however and could require several parts of the program to be rewritten to allow it to work, so it may not be the most practical enhancement to make.

Another feature i would add to my Tic Tac Toe game would be different shaped game boards. The traditional game board for Tic Tac Toe is a 3 by 3 grid but i would like to add the option of game boards of different sizes with different rules to win. For example a 5 by 5 game board requiring a user to place 5 counters in a row to achieve a win. This could be expanded upon to create different rule sets for the game to allow it to be played in a new way. One rule set could be the introduction of a points system within the game. Each player would be assigned an amount of points dependent on how they win the game. For example if a player wins by getting three marks in a row diagonally this would award more points than 3 horizontally. This could also lead to the inclusion of a leader board where each users high scores are kept. Introducing this could lead to a more competitive aspect of the game and would give players a reason to keep playing.

In terms of the program code itself there are a few general enhancements that could be made to improve the program. I would continue to improve on the games memory consumption and would continue to try and reduce the amount it uses. Finally i would also improve my replay function to also display any undos or redos made in the game as currently the replay function does not display undos and redos. With this the replay function would be more accurate.

4 Critical Evaluation

As a whole i believe my program turned out well and matched the initial specification however some parts work better than others. I believe that the base game works very well in my program as it is not memory intensive and allows the user to play a game without fail each time. I also believe my replay function has been implemented well as when the replay command the user is given a turn by turn breakdown of the game in an easy to read fashion. It displays a message of what the move was that was made and which player made that move, it then displays the game board with this new addition added. I then implemented the Sleep function to make each turn play one by one instead of all at once. This has resulted in a simple replay function that is easy to read and understand. However as the replay function does not keep track of moves that are undid or redid it does not work well for games that have several undos and redos. My game

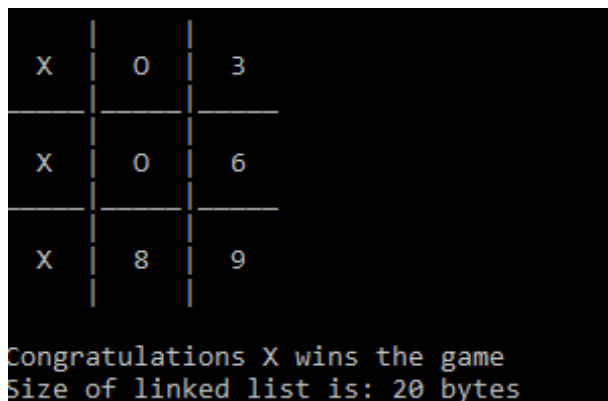
also allows the ability to play several games in a row without any memory issues so the user can play as many games as they want in a row. The memory allocation in my program also works well and saves a lot of memory that would have otherwise been wasted. I do this by dynamically allocating the memory for my linked list as required using the malloc and memset functions as shown below:

```
1 *list = (struct node *) malloc(sizeof(struct node));
2 memset(*list, 0, sizeof(struct node));
```

To test how much memory my linked list uses I wrote a simple function to count the number of nodes in the linked list then multiply that by the size of the data type the linked list holds which in my case is an integer.

```
1 int countList(struct node **list)
2 {
3     struct node *temp = *list;
4     int count=0;
5
6     while(temp) {
7         count++;
8         temp = temp->next;
9     }
10    int size = count * sizeof(int);
11    return size;
12 }
```

This allowed me to test the size of my linked list depending on how many elements are present in it once a game has been completed. For example in a game where there were 5 entries into the linked list the size was 20 bytes:

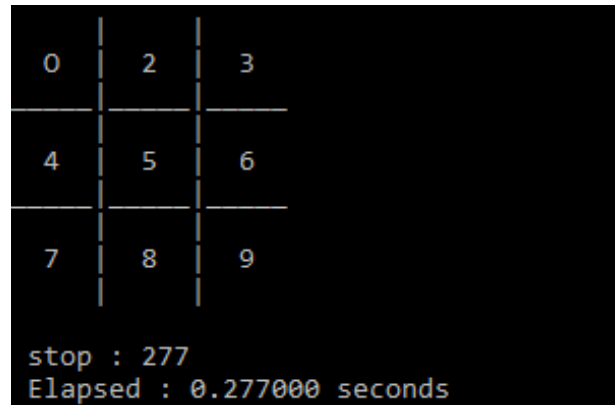


So each node in the list requires an additional 4 bytes of data to be used as another integer value is being added into it.

I also tested the time taken to elapse 1 turn of the game using the following code which was provided in the second lab of the module:

```
1 printf(" start : %d \n", (int) (t = clock ( ) ));
2 printf(" stop : %d \n", (int) (t = clock ( ) -t));
3 printf(" Elapsed : %f seconds \n", (double) t / ←
    CLOCKS_PER_SEC );
```

I placed the start line of code before my tictactoe function was ran and the other two statements after the function had been run. However the time taken for a turn to progress is dependant on how quickly the user selects a place to put their mark. If the user was to enter a number as soon as the option is given the timing is as shown below:



There are however some areas of my program that don't work as well. My undo feature has a bug that I have not been able to solve which occurs when the user tries to redo after undoing to the initial board state. When redo is entered it will start from the second value entered and not the first value. This is a flaw with my undo function that I would improve on and fix in the future. Another area I feel could be improved is the memory management of my program. Even though I have put some work into minimizing memory usage as I mentioned above there are more things that could be done to reduce the memory usage of my program but for a project of this scale the memory I could save would probably end up being negligible but if this was a much bigger more expansive project then this would become more of an issue and my memory management would have to be improved.

5 Personal Evaluation

During this coursework I have learned a lot about using data structures and algorithms in the C programming language. I feel I have gained confidence in implementing a project of this scope using C and I can now implement several different data structures with a general understanding of how each one works and how I would use them to tackle a problem like the one I have done for this coursework. The first problem I encountered while doing this coursework was deciding upon which data structure to use to store the player moves as there is plenty of viable options. After carrying out some research and completing the labs for several of the data structures I decided to use a doubly linked list as I believe it was the most practical choice for completing this kind of problem as the ability to go backwards and forwards between nodes was incredibly useful for implementing the undo and redo features outlined in the specification. The hardest problem I faced during this coursework was implementing the undo/redo function. I struggled to find a way to loop through my linked list to the desired turn that the user had undid or redid to. I eventually found a solution by creating a variable to count the number of turns that had taken place, this allowed me to find the specific turn the user was on by incrementing the turn counter depending on if an undo or redo has been made, I then show the board state for that desired turn. I also had some problems trying to read the game data saved from the file into the program to implement the replay feature. I was trying to use the fscanf function to read the data into an array but after hours of trying it still did not correctly read the data from the file. I solved this problem by

changing to using fgets to read from the file which i managed to get working relatively quickly without any issues. This is a lesson i learned several times during this coursework and I'm now more prone to look up alternative approaches for carrying out tasks where as before i would stick to the way i knew or the first way i discovered.

Overall i think i performed relatively well with this coursework and have implemented not only a working game but i have also implemented the additional requirements laid out in the specification. However if my time keeping skills had been better i would have liked to have gone beyond those requirements and implemented some of the improvements i listed in the enhancements section of this report. This is something i would improve on if i was to undertake this coursework a second time.

References

<https://stackoverflow.com>

<https://www.tutorialspoint.com>