Edinburgh Napier UNIVERSITY

Coursework Report

Guillonneau Alexis
40404292@live.napier.ac.uk
Edinburgh Napier University - Algorithms and Data Structures (SET08122)

Abstract

The objective of this coursework is to demonstrate the understating of both theory and practise in relation to the content of the Algorithms and Data Structures module. The task is to implement a text-based Tic-Tac-Toe game using the C programming language.

Keywords – Report, Tic Tac Toe, C, Data Structures

1 Introduction

The core goal of the implementation is focused on the choice of data structure and algorithms needed to implement the game that I made. In this report, I will talk about the solution I took to implement the different features. Furthermore, I will cover the other features that I did not had time to implement. In my program, is it possible to change the size of the board, for example 40x40. A feature allow the users to save their game of continue it later for example. With this feature comes another, the one which allows to continue the games which has been stopped.

2 Design

Firstly, I created a data structure for the players. In this structure, we have a string (which is represented by an array of characters in the C language) for the name of the player, also a single character define the sign the player is using, a nought (O) or a cross (X). We do not need further information of the users in this minimalist game.

Secondly, we needed to create a new data structure in order to store the coordinates of a node. This structure named "Coord" in the source code is composed by to integers for both x and y coordinates.

Thirdly, the most important data structures represents all the nodes that composed the game, the name of the later is "Node". In this implementation, I chose the double linked list non-circular. It is the simplest way to represent what I needed, namely the previous and the next node and a way to store the data and the coordinates of each node. The first type in this data structure is a character called "data" which represent the value of the node. It is just to know if there is a cross, a nought or nothing on the node. There is a type that represents the coordinates of the node, so I used the data structure "Coord" created earlier. Then we have two pointers to link the current node the next and the previous node. Considering that we do not need the first and the last node to be linked, on of their pointers are set on 'NULL', that is why is it a non-circular double linked-list. On the current state of the program, a single linked list would have been sufficient but in order to implement the undo and redo features, I needed a double linked list (figure 1). An other

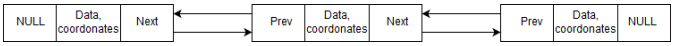

Figure 1: **Double linked list non-circular** - Representation of the double linked list non-circular I am using in the game

way to represent the nodes of the game is by using a hash map of each node where the key would be the coordinate or the id of the node and then store the data. But it is not a convenient solution for doing the undo/redo feature.

For saving the game, I store the detail of the current game in a file name "data.txt". The format is 'name1 name2 SizeOfBoard;x y data;...;'. On the latest version of the program, this feature does not save the name of the two players because when I read the file in order to continue a game, I encountered problems while reading and treating the different characters.

This is one of the four main functions in the source code. The second one is the modification of the board size.

The third one is to save a game. In order to do that I create a function that write all the details of the current game on a txt file. When the users play a game, they can save it whenever they want and also continue directly after the saving, I needed a way to update the part on the data file corresponding to the current game. I found only one way, it is to copy the data of the file on a temp file without the last game (the current game) and then copy the data of the temp file on the real data file and then add the last information of the current game.

The last one is the playing feature. It is in this function that the players are initialized. Then I used a while loop if the condition where if there is a winner or if the game is full the game is ended. To check if there is a winner, i use a algorithm that check first the horizontal lines, then the vertical lines, the first diagonal and the second diagonal. For each check the complexity is O(N^2). This function uses a lot of memory and time but I did not find another algorithm in order to check if there is a winner or not.

3 Enhancements

In order to implement the feature that allow undo and redo, a new data structure would be needed. It would take the form of a stack of all the move that the players have one. This stack would be a double linked list. If the player and to undo a move, we just need to modify the address where the pointer is fixed on. We use the same idea for the redo feature. It would be a big change on my implementation because I would no longer need one node for one box of the board. It would be just a stick of the different moves. It is the big improvement that I would do if I had more time.

The second thing is to implement a sort of artificial intelligence. It would use a search algorithm based on tree like the min max algorithm. It would choose the best option in order to win the game or at least make a draw. This implementation requires a lot of time even if I know the theory of the algorithm and I am familiar with this kind of algorithm in the programming language Python.

I would also improve the storage of the different games. With the implementation that I talked about earlier, the only things that would be need is the succession of moves and not the state of every box in the board. The size of the file would be shorter as well as the time and memory taken by the different writing and reading algorithms.

I would also change the text based interface because when you play it is not very convenient. Indeed, at the beginning of every move, the program ask to the user if he wants to save the game or not.

4 Critical Evaluation

The feature that allows to play play a game as well as the adaptability of the board size work well. The saving feature works a bit poorly in my opinion because there is an issue if the players name and also because for each game it take the length of the two names + 4 + 6 * $boardSize^2$ characters. So the space taken increase significantly with the board size. Or it should increase by he number of the moves.

There is also a poor error treatment during the writing and the reading of the '.txt' file.

5 Personal Evaluation

I already knew the C programming language but I ave not used it on the last 2 years so it differ a lot with the language I am very familiar with, Python. In the C programming language, we need the handle a lot more thing like the different data structure, the allocation of the memory, etc. I went through a lot of problems during this coursework because I got to remember all the things that I learned 2 years ago. But thanks to the different lab session and the documentation that I found on the web and in the book[1], I solved a lot of problem despite the time it took. I learned during this project to choose carefully the data structure that will be used in the program and the different algorithms that I could use to resolve on particular problem. i also realized that many things that we can do in others language like Java, Python or even Go is a lot difficult to do in C.

References

[1] M. Olsson, *Modern C Quick Syntax Reference*. Hammarland, LÃďnsi-Suomi, Finland: Apress, 2019.