ALGORITHMS & DATA STRUCTURES

Tic-tac-toe

2019

EDINBURGH NAPIER UNIVERSITY
Bogoslava Dyankova
Matric. No: 40323952

Introduction

The objective of this project is to create a text-based Tic-Tac-Toe (noughts and crosses) game, programmed with C. This adaptation of the game includes all basic principles as the original, as well as containing other advanced features which allow the players to undo and redo their moves, as well as see how the previously played games have gone step by step. The aim of this report is to explain the different data structures used throughout the project to ensure that it performs to the required standard along with why these structures were used.

Design

The design of the game is divided into different data structures.

The basic data structure of the program is created using a two-dimensional array. The 2D array in the C language is also known as matrix. It can be represented as a table of rows and columns. The difference between 2D and a normal array is that you must always specify the second dimension even if that's done during the declaration.

An advantage of using arrays in programming is that they represent various data elements of identical type using a single name. The number of elements that are going to be stored in the array should always be known. Two dimensional arrays can be useful for organizing smaller groups of data within an array. They can be really helpful if we want data to be stored in matrix form. As we are using the same type of data, arrays are a convenient way for saving data of the same data type with same size.

The undo and redo features, which are an extension to the original game, are built using arrays. The usage of arrays is more beneficial when it comes to a locked board (3x3 for the Tic Tac Toe) as it has a set size in memory which makes it easier to store and search data. Repeating the arrays using their index is quicker than using different methods like linked lists.

A 'dummy array' is created for the Replay game function, which serves as an empty array, ready to be filled with the data from the previous games and display how they went turn by turn. As the game goes, every single move is recorded step by step so it can be shown afterwards in the Replay if requested.

The game starts checking for a winner after the 5th turn, as there cannot be one before that. That is done by a 2D array, checking every row, column and diagonal for three signs of the same kind (Crosses or Noughts). If the validation returns such a result, the program flags a player as a winner. If it doesn't find any of the combinations, the game continues and the array keeps on checking after every turn.

Many validations were made in the code just so random program failures are avoided. Plenty of 'if – else' statements were used for values in the arrays so there is low possibility of a player to crash the game.

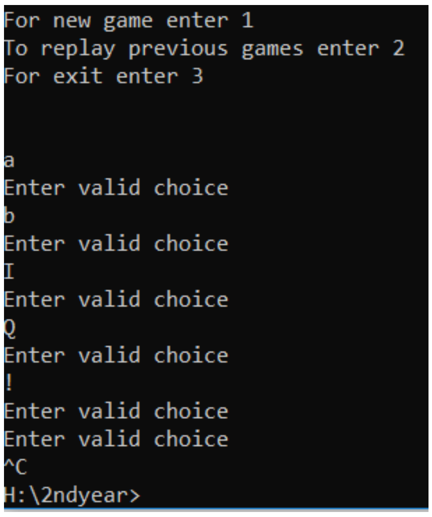
Enhancements

Many ideas come to mind when such a simple game as Tic-Tac-Toe is given as a topic. One of the changes would be putting different levels of difficulty. As an example, a timer would be put on every turn (5 seconds) so it requires faster thinking and bigger attention. For this to work properly, on this difficulty the Undo and Redo options should be removed. Another level option should be the size of the game field – instead of 3x3, you get to choose how big you want it to be.

The players would be able to type in a nickname, which saves into a database so whenever one wins a game, it goes into a leader board under their nickname. Thus, nicknames should be unique.

Critical Evaluation

All the normal features work well enough for a game played in the command prompt. The validation is done well enough so most unexpected crashes are avoided. The undo and redo functions are well implemented as they fetch a specific type of data, which if not found, the game requests a proper input. Another feature which works well is the Replay game option. It outputs the number of games played and the game itself shown turn by turn, showing which player won or if it was a draw at the end.



```
For new game enter 1
To replay previous games enter 2
For exit enter 3



a
Enter valid choice
b
Enter valid choice
I
Enter valid choice
Q
Enter valid choice
!
Enter valid choice
Enter valid choice
^C
H:\2ndyear>
```

The validation for ignoring all types of valid choices is not working properly. If the player types in a normal character (a-z, 0-9, symbols) the game will ask for a valid choice. However, if one enters a combination of two buttons (i.e. Ctrl + C) the game just exits and needs to be run again as shown in the image.

An element which could be improved on is the Winner conditions. It is currently checking every type of combination in the array if there is a winner. Despite the code looking perfectly, it looks unprofessionally done.

Personal Evaluation

The game was pleasant to make and really profitable for expanding my knowledge. Debugging the program was probably the biggest challenge I have faced as my debugging skills aren't as good. The whole coursework helped me improve my programming skills. I overcame the challenges I faced mainly by looking up in the internet how to fix a huge amount of the errors I had. The work I have done is satisfying for my skills, although one should always improve and aim for more. I am overall pleased with what I have done, however, if I had spent more time on the coursework, I might have been able to achieve more.

References:

https://www.tutorialcup.com/cprogramming/array-advantages-disadvantages.htm