Algorithms and Data Structures Coursework 1 Report

Jonathan Binns

40311703

Introduction

Describing the problem and giving an overview of features

The main objective of the task is to create a functioning naughts and crosses game, while displaying our skills in implementing different data structures and algorithms to act on those data structures. In order to create naughts and crosses I need to represent the different aspects of the game, these are the game board, the players, the pieces and the positions the pieces are played at. These can be done with different data structures and how I will represent these will be discussed in the design section.There will also be some extra features implemented in this game, the first is the ability to display instructions so the user knows how to play the game. The second is that I will decide which player goes first by ‘flipping a coin’. The third feature is adding the ability to undo and redo moves made so players can fix any mistakes they make. Finally the last feature will be the ability to record games so they can be replayed at a future time.

Design

Explaining how you designed and architected your software playing particular attention to the algorithms and data structures used

What I need to represent/how I represent them

My design approach centers around what needs to be represented in this game and how I could go about representing them.

Board/1d array

The first, and most important, thing I need to represent is the game board itself. I will use a one dimensional array of nine spaces to do this. The main reason for this is that it’s the simplest approach which means I can build my functions around it easily as I can calculate which positions in the array equate to each position in a board. It also helps to minimize the amount of loops I will need when dealing with it, as there is only one ‘row’ I only need one loop to go through each element.

Pieces/chars

The next thing I need to represent is each piece played by the users. For this I will be using a char, either an ‘X’ or an ‘O’. This means I can make the array that represents the board an array of chars. My reason for representing moves like this is that a naughts and crosses game is typically played by writing an ‘X’ or an ‘O’ so by representing each move with those characters it allows the game to look as normal as possible.

Positions/ints

Each piece also needs a place on the board to go, to make sure they go in the intended place I will represent the board positions as ints. These ints will correspond to a position in the array meaning that an insert function can take in a char and an int positioning the char, representing the move, in the place the int dictates in the board array. This is why I chose a one dimentional array to represent the board as it makes adding moves as simple as possible as each position in the array is only accessed by one number, allowing moves to be inserted quickly as there is no need for a loop.

Players/chars

Each player will be represented as a char, just like their moves. This can be kept track of easily with a for loop that changes which player’s turn it is to enter a move.

Undo/redo feature/stack

My undo and redo move features will be represented by a stack. My plan is to have a stack of boards and add a board after a move has been made. This means that when calling the undo or redo feature the top board on the stack just has to be made equal to the board that is currently in play. This means that there is no need to represent moves as individual structs and it takes out any errors in terms of people wanting to undo a move that changes a space on the board that is not null.

Replaying games/stack/.txt

Finally to allow the user to replay games at a future date I will add the stack of all boards to a .txt document at the end of each game. This is the best way to represent this as I already have to keep a stack for undoing moves so printing this to a .txt document is not a hard extension of that.

Enhancements

Describing the features that you would add or improve if you had more time

Having the menu as a function

The first thing I would improve in my program is having my main menu become a function rather than a set of print statements in the main method. To do this I would pass my mainMenuChoice integer through as a parameter that gets changed based on the user input in the command line. Then this would correspond to the if-else statements in the main method. I would also do this with my game menu and pass the gameMenuChoice integer as a parameter that would get changed to decide what the next thing to do in the game is.

scaling the board up to connect 4

A second thing I could add to my program is give it the ability to scale the board up to a 4 by 4 or maybe a 5 x 5 grid. To do this however, I would probably have to re-write most of my code so that the board was a two dimensional array rather than a one dimensional array. I would do this because I could re-write functions like checkWinner() and display() to use loops rather than hard coding them to work with a 9 space array. This means that I could use all of my functions for any sized board, keeping the repeated code to a minimum and also keeping the main method concise.

a.i.

Another feature I could add is an ‘a.i.’ for the user to play against. To make this I could start by having an algorithm to loop through the board and randomly pick a free space to play a move. This is the most basic form of an a.i. I could think of. I could continue to make it more sophisticated by using if-else statements to tell it to block any rows or columns the user is close to completing. A final improvement to the a.i. could be choosing a strategy out of a few possible ones to try and win the game. The strategy would just be a set of moves to be played however that would allow the a.i. to play aggressively instead of always trying to play defensively.

heat map

The final feature I would add if I had more time is a ‘heat map’ which would show the number of people who played in each square after the first move has been played. This could be implemented by having each starting move and second move recorded in one txt file and each time someone plays a second move a counter would increase on the positions that were played. It would be displayed by showing a number in each position in the board once the first move has been played. This could be helped to improve my a.i. by having it always choose the most popular move that has been played after each move the user makes. However this would take a lot of time, and would continually slow down the game as the a.i. would get more and more complicated with more if-else statements to make and for it to get caught up in.

Critical evaluation

Explaining the features that you feel work well, or work poorly, and why you think this. You should support your evaluation with experimental results.

Overall I think my functions work well, I achieved this by making sure they were all as simple as possible to keep processing time down. I also did this by, where possible deciding to hard code all of the possible combinations for my array than using a loop, which then could be scaled.

Display()/works well

My display function was one of the first I wrote, and I think it works well. This is an example of me ‘hard coding’ the function rather than using a loop. This function takes in an array of characters. The first three are displayed on the first line, the second three on the second line and the last three on the bottom line. To make it look like a board I added characters such as | and – to separate each square (Geeks for Geeks. 2019).

addMove()/works well

The second function I think works well is my addMove() function. This function takes in an array of characters, asks the user for a position and a move to be played. Then it checks that the position in the board that they have chosen is null. After that it calls my insert function which takes in a char array, an int and a char. The array for the board, the int for the position and the char for the move. My insert function just inserts the move played at the position specified in the array. It also checks that the character is a capital X or a capital O. After adding the move the user wants to make to the array addMove() then calls my display function. I think this function works well as it does everything needed to safely add a move within a small number of lines. It also works very fast with little or no input lag or time taken after the user types in a move.

checkWinner()/could be improved

My check winner function is one of the functions I could improve given more time. This function takes in, a char array, for the board, an int, for the winner variable which becomes 1 when a winner is found, and finally a char for the winning player. The function uses a series of if-else statements to go through all the possible combinations of a win by looking at which positions in the array are equal to which character. If a winner is found the winner integer is set to one, breaking the loop in the main method, and the winning player character is set to which character has matched the positions in the array. If I had more time I could improve this by using a set of loops to check for the rows, and columns and only two if else statements for diagonal wins. This would also mean id have to re-write my array as a two dimensional one as looping through my one dimensional array would take roughly the same amount of code and would take a little longer to execute.

checkFullBoard()/works well

The next function I think works well is my check full board function. This function takes in a char array and an integer, the array represents the board and the int represents a variable that becomes 1 when a full board is detected. The function loops through the board and for each position that is not null it increments a counter. Once the loop has gone through all elements in the array there is an if-else statement that checks if the counter is 9, in which case there is a full board. Otherwise the full board variable is kept at zero. The full board variable is used in my main method to stop asking the user to input moves once they have filled the board. I think this function works well as the loop will at a worst case go through 9 elements, this keeps it quick to run as 9 elements is not a lot. Alongside that at a worst case it is incrementing the variable 9 times but at maximum it has to do this once a game, as the board is full.

stack/works well

I think My stack functions work well overall. The first function I wrote was the initialize function. This takes in the stack and sets every element to be null. This taught me a lot about a stack as a consept and two dimentional arrays. The next function I wrote was the display function, this takes in a stack and loops through each ‘layer’ and calls my display function to display the elements in that array. This was mostly used for testing, to make sure that my initialize function worked and to test my push function before I started work on my pop function. This function is quick as it only have to loop through 9 elements, and because my display function doesn’t include any loops each array can be printed as quickly as possible. The next stack function I wrote is the push function. This takes in a stack and a char array. It then increments the value for the top of the stack and loops through the char array, representing the board, and sets it equal to the current layer in the stack. This is also very efficient as it only has to do one loop of nine rather than one for the layer and one for each element in that layer. Finally I wrote my pop function, this takes in a stack and a char array. This function is slightly different to a normal pop function as instead of returning the value of the top of the stack it replaces the game board with that array. The function first decrements the value for the top of the stack. It then loops through the elements in that layer of the stack and sets them equal to the elements of the board in play. It then calls my display function to show the board. Overall all of my stack functions are concise and efficient with the biggest loop only needing to be done once a game. Asides from that I make sure to keep loops to a minimum and I call as many other functions as I can to keep the main method as quick as possible.

read/write from file/works well, takes a second to get time

My final two functions are my read and write file functions. The write file function takes in a stack, it then creates a pointer to a file and gets the time, in seconds since the first of January 1970. The function then creates a file called game(time).txt and opens it in write mode. The function then moves through each layer of the stack and writes them to the txt file in the same format that they are printed. I couldn’t use my display function here as that uses printf whereas I needed to use fprinf, this doesn’t effect efficiency but if I had more time I would have liked to try and have a function to actually write the arrays to the file rather than using near identical code twice. After the function has looped through all layers of the stack it closes the file. Next my read function this doesn’t take in any parameters. It first creates a pointer to a file, it then asks the user to enter the name of the txt file that holds the game they want to replay. It then opens the file in read mode, and after checking that the pointer isn’t null, loops through all of the characters in the txt file and prints them to the screen. After that it closes the file. The only problem with these functions are that sometimes the write function can hang for a second or two when generating the time. Asides from that both are efficient and do their job quickly.

Personal Evaluation

Reflecting on what you have learned, the challenges you faced, the methods you used to overcome challenges and how you feel you have performed.

References

Geeks for Geeks. (2019). Implementation of a tic-tac-toe game. Accessed On: 25/03/19 Retrieved From: <https://www.geeksforgeeks.org/implementation-of-tic-tac-toe-game/>