# Algorithms & Data Structures: Noughts and Crosses Game

## Introduction

The aim of the assessment is to produce a functional game of noughts and crosses. The game will be developed through the use algorithms and data structures and will allow to players to interact with each other. The game created has as functional board, two players (represented by 'X's and 'O's) and a grid for them to play. The noughts and crosses game will end when either a player gets three in a row or if 9 moves are played (then it is a draw). It will display the winner and exit the game. In addition, after the game has been played it will display all the previous move of both players.

## Design

To develop a functional noughts and crosses game the main data structure used was an array. As arrays are already features in the C language it made sense to apply it to this project. Arrays are Linear (Sequential) data structures, meaning that the data is stored in a linear order, for instance, in a row (Mehlhorn & Sanders, 2008). A one-dimensional (1-D) array was chosen for this assessment as they are easy to work with and stores data in a sequence. To create the graph the row was split into three rows of three columns for the game within the showBoard() function. When the game starts, all the places in the grid are set to empty within the initialiseBoard() function. The showBoard() function is used to specify the index of the data in the grid and how the array should be displayed when the game is run. So, when a player enters a number the playerMove() function stores their token (i.e. either an 'X' or an 'O') in a specific index in the array, which is later used to check for a win using the winGame() function. The winGame() function checks the board each time for a winner and validates the data entered. For instance, a player cannot place their token in a place that is already occupied, and they must enter a number between 1 and 9. Each instance of how to win them game is stored in the function and checked within the main method- using a nested if statement. The size of the array is known in advance and is set to hold a maximum of 8 characters. This means that the array used could also be described as a bounded data structure (Mehlhorn & Sanders, 2008) which would make it suitable for allocation on a stack (Stevens, Koch, Walter & Winkler, 2004).

A stack is a linear data structure which can be used in C. Stacks follow a particular order and can be only accessed at the top (Adamson, 1996). This means that the last item stored is the first item that can be used. For the game it was used to store each of the moves made by each player (or the boards of each move). This would make a stack suitable for a noughts and crosses game as it could be used to "undo" players actions by removing their previous move or "redo" their actions

(Adamson, 1996). For the implementation of the stack a two-dimensional array was used. The stack was created using a struct which creates the 2-D array and sets the top of the stack. The array for the stack was used to represent the rows and columns for each board (each board being stored every time a move is made). The push function is used to store each move (i.e. the board) onto the stack which is later accessed using the pop function. Once the game is over the pop function prints out each board in the game allowing the players to see a replay of their game.

## Enhancements

An enhancement which could be fixed through is that currently if the user inputs anything other than a number, such as an alpha character, it continues to loop and must be broken. This could be fixed with a loop to output an error if there was an alpha character entered. Another feature that could be improved would be to update the pop function. Currently, the pop function prints out nine boards. However, if the game is won in less, it'll print out the boards stored in the stack as well as incorrect boards with random outputs. This could be resolved by implementing a loop to loop through all the current boards in the stack and to exit when there are none left.

Enhancements that could have been added to improve the effectiveness of the code would be to implement an undo and redo feature. As a stack has already been made, it would require a new function to move back through the stack to undo the "moves" of a player. The same in reverse for the "redo" of moves. There is also opportunity to implement a computer interface, so instead of requiring two separate players, it would only require one player to play against the "computer". This could be done initially for the "computer" to find a random empty space on the board to place their token. Also, a menu would be a convenient feature to improve the user interaction.

## Critical Evaluation

Overall the functions in the game altogether provide an effective noughts and crosses game with areas that could be optimised. For instance, an improvement that could have been worked upon would be the way I implemented the getMove() function. Currently, the function has multiple if statements for each way a user could input their move. I believe that this could be improved using a loop instead of having a series of if-else statements. However, this method is efficient as it allows me to validate the input to not allow the player to place their token in a space already taken as well as only allowing the numbers 1-9 to be inputted (within the if-else statements)

The showBoard() function, I believe, works well as a feature to represent the one-dimensional array. This is because the function shows how the array should be indexed and where the players tokens are stored. It also displays the board as a standard noughts and crosses game appears, which will be

familiar with most players. The idea for the showBoard() function was inspired by Sourabh Somani's board function in their own version of the game (Somani, 2014).

Another function used in the game is the winGame() feature. I believe that this feature could be improved upon as I have had to repeat each scenario a player could win the game as either the noughts or crosses. More research could have perhaps found an alternative method to implement, such as a loop to find the 3 moves in a row, column or diagonally. This function is used within the main to compare each current board and will end the game if a win condition is met. The idea for the winGame() function was inspired from a Youtube tutorial (NVitanovic, 2014).

At the top of the code, I state that the player to always start as 'X'. Then, within the playerMove() function I switch. So, if the player before was 'X' it will switch to player 2 or the 'O's (the same vice-versa). Whilst this is an effective function, a possible consideration would be to allow the player to choose which token they would like to be.

For the implementation of the stack I used push and pop functions. The push() function, being inspired from lab 3, I believe is an efficient method to push the current board to the stack. It includes two for loops which go through each row and column in the two-dimensional array. The top of the stack (i.e. the top row) is then set to equal each board (row) in the array. The top of the stack is then incremented each time until it reaches the MAX amount.

The pop() function is used to go through each element in the stack. Each time it goes through the stack it decrements the top, allowing the pop to go backwards. The for loop allows the stack to go backwards through the array, meaning that for each row in the board array, it points to the top of the 2-D array of the stack. This is an effective method to look through the stack.

## Personal Evaluation

On reflection, I believe that the game created works and performs well. It meets all the basic functionality set out in the assessment with the additional feature of replaying the moves of the previous game. The main challenge faced was implementing the stack. By using the tutorials, I was able to get some of the main initialisation and push functions made however, it was challenging to implement an appropriate pop function. This required some research and going through the labs to get a better understanding of how to create the function and implement them in 2-D arrays.

By building this program it has allowed me to put into practice skills learnt from the labs. I have learnt more about different types of data structures, particularly 1-D and 2-D arrays. A challenge that I initially faced when building these data structures was creating appropriate functions and how

to implement them in the main method. This was overcome by taking time to plan how each function would work with one another to create the game.

Therefore, I have developed the skills to enable to produce a working noughts and crosses game, so I believe I have performed well. Although there were challenges, I was able to overcome them through the skills developed in the labs with further research done. This has allowed me to develop a better understanding of data structures and further develop my skills programming in the C language.

## References

Adamson, I. (1996). Stacks and Queues. Data Structures And Algorithms: A First Course, 27-56. doi: 10.1007/978-1-4471-1023-1_2

Mehlhorn, K., & Sanders, P. (2008). Representing Sequences by Arrays and Linked Lists. Algorithms And Data Structures, 59-79. doi: 10.1007/978-3-540-77978-0_3

NVitanovic. (2014). C++ Tutorial 15 - Making simple Tic Tac Toe game (Part 1). Retrieved from https://www.youtube.com/watch?v=xwwl8TgkwgU

Somani, S. (2014). Tic-Tac-Toe Game in C#. Retrieved from https://www.c-sharpcorner.com/UploadFile/75a48f/tic-tac-toe-game-in-C-Sharp/

Stevens, M., Koch, M., Walter, J., & Winkler, G. (2004). Bounded Array Storage. Retrieved from https://www.boost.org/doc/libs/1_46_1/libs/numeric/ublas/doc/bounded_array.htm