Data structures and Algorithms Coursework

# Introduction

The task that I have been given is creating text based Tic-Tac-Toe game by implementing several of data structures and algorithms in the C programming language. The game must have a game board, players, pieces and positions. The game must record history of play and enable earlier games to be automatically replayed from this record. The game must support undo where once a move is made, you can undo the move to the game state before where players can undo moves to initial game state. The game should also have a redo feature that allows redo moves which been UN done. The features that the Tic TAC Toe game has a register and login where users can register an account by setting a username and password. The password is encrypted then the encrypted and username is written to a file. The login feature is where user can login into the game as long the username and password equal to username and password in a certain line of a file. The other features are a multiplayer game where two humans can play with each other by entering the square number that a user want to position their mark on until a player wins or there is a draw. The game moves is then replayed from initial state to end state in a way where each position is placed every time a user presses a key until finished state. A player can then decide to play again or go back to the menu. There is also two other types of games where a player can play against a computer. One game is where computer makes a random move every time it is turn. The other version is where a minimax implementation is used to allow a computer to see all of the possible moves and chooses the best move every time the computer makes a move. There is two replays search system. One is where a user can enter a game id key and a certain moves from games that have been played would be replayed subsequently. Another is where a user can enter a game id key and certain moves from a game from among the games that certain user have played in the past.

# Design

### Tic TAC Toe Game – Multiplayer Design

I have used a 1D array of chars for the game board where each char represent a square of the board which initially would have ‘1’, ‘2’, ‘3’, ‘4’, ‘5’,’6’,’7’, ‘8’, ‘9’. Depending on what square number, that a player have positioned a piece on the game board, the string would be replaced with a certain piece, either a nought(‘O’) or cross(‘X’), depend on what player have positioned a move. I have a used a 1D array because it enables random access of elements which makes accessing elements by position quicker every time a player makes a certain move. It also have better cache locality compare to data structure like linked lists, hash map, stacks and queues which improves performance of the game. I have used a 1D array instead of a 2D array for game board because the total bytes of a 1D array just depend on size of char \* size of array where as the total bytes of a 2D array just depend on (size of char) \* size of first index \* size of 2nd index which mean 1D array takes less memory space compare to a 2D array. Also, the 1D array can be received in pointer, sized array or a unsized array whereas the received parameter must define rightmost dimension of a array which makes the game board easier to be implemented in a 1D array compare to being implemented in a 2D array. I have implemented the game board by using a Array instead of linked list because Array supports Random Access, that indicate elements can be accessed directly using their index such as when a player decide position a cross piece to square 7, square[7] can easily change from ‘7’ to ‘X’ where as Linked list supports Sequential access, that indicate accessing any square of the game board, we have to sequentially traverse the complete linked list upto that square. Therefore, to access nth element of a linked list, time complexity is O(n) where as accessing elements within array is fast with a constant time complexity of O(1). I have implemented the game board in array instead of a stack because in a array, the objects is arranged in a way so it can be accessed at any time in a random fashion where as in a stack, the objects arranged in a way where it can be added and removed from one end only so when a move is un-done, it is more easier for 5th square of the game board to be set from ‘X’ to ‘5’ when the game board is implemented in a array compare to when it is implemented in a stack. In a stack, the elements can be added or removed within a LIFO order indicating the last one in is the first to be accessed and first one in can be accessed last where as array is a random access operation and everything ggets down to the start of array. Insertion and deletion takes place within any position. This mean that a array implemented game board makes it much easier, faster and efficient for the moves to be replayed compare to a stack implemented game board.

I have used a stack to implement the undo and redo function. A move is undo every time user types in ‘10’ and a player can redo a move by typing ’11’. I have created two stacks which is a stack that contain all of the moves which is called ‘undoStack’ and a stack that contains all of the moves that been un-done, which is called ‘redoStack’. Every time a player makes a move, moves in the ‘redoStack’ is destroyed and a move is pushed into the ‘undoStack’. When a player (either player 1 or player 2) undo a move, a move is popped from ‘undoStack’ and pushed into the ‘redoStack’. The game is then set to the previous game state. Players can keep undoing moves from current state till initial game state where the game board is empty. When a user redo a move, the the un-done move will be re-done where players can redo until all undone moves are back on the game board. This happens by popping a move from ‘redoStack’ and pushing into ‘redoStack’. When a player makes a move, player can no longer redo moves that have been un-done. I have used a stack instead of array to implement this feature because a stack is a linear data structure shown by a sequential collection of elements in a fixed order where a array is a collection of related data values called elements each identified by a index array which mean is more faster for user to undo and redo move by using stacks than implementing both features with array. Also, stack elements can be added or removed in a LIFO order meaning last one in is first to be accessed and first one in can be accessed last where as a in a array, it is a random access operation and everything gets down to start of array. Insertion and deletion takes place in any position. This indicate that it is more faster to undo and redo moves by using a stack instead of a array. I have used a stack instead of a queue to implement the undo and redo moves as it is simpler to implement both features using a stack compare to a Queue. Also, in a stack, same end is used to insert and delete elements where as in a Queue, on end is used for insertion such as rear end and another end is used for deletion of elements such as front end which mean stack makes more faster for players to undo or redo moves compare to a Queue.