Algorithms and Data Structures Coursework Report

Qasim Imtiaz
40274561@napier.ac.uk
Edinburgh Napier University - Algorithms and Data Structures (SET08122)

1 Introduction

This report describes the design of a Tic Tac Toe game by describing the algorithms and data structures been used for each feature and justifying it by analysing the performance and time complexity for each data structure and algorithm. The report also describes possible enhancements in the future, critically evaluate what features have worked well, what features have not worked well and the reason for it. The report finally draws personal evaluation of the game which states what I have learned, challenges I have faced, techniques used to overcome these challenges while implementing the game and how I have felt I have performed overall. Tic Tac Toe game has a register and login feature where each user can register account and login into the game where the user password is encrypted. There are three types of Tic Tac Toe games. There are a Multiplayer game where two humans compete with each other, a hard version of Tic Tac Toe game where a human player plays against an A.I that uses a minimax algorithm to choose the most optimal move and a beginner version of Tic Tac Toe game where player can play against a computer that randomly chooses a move. In each of these games, players can undo and redo moves. At the end of every game, the moves subsequently replayed from initial game state to final game state. There is also a feature where players can search for replays of a particular previously played game by typing in the game id and feature where a player can see replays of a particular game that specific player has played before by typing in the game id.

2 Design

The tic tac toe game is played on a 3x3 game board by two players, who take turns where the first player moves with a cross and second player move with a circle. The player who has formed a horizontal, vertical or diagonal sequence of three marks wins. When the game board is full, and no one wins, it is a draw. The game board implemented by a 1D array of chars where each char represent a square of the board which initialised with values "1", "2", "3", "4", "5","6","7", "8", "9". Depending on what square number, that a player has positioned a piece on the game board, the string replaced with a certain piece, either a nought("O") or cross("X"), depending on what player has positioned a move. The Game board implemented as a 1D array instead of a 2D array as 1D array stores data in a list whereas 2D array stores data in a row-column format, Therefore, it is easier to implement and faster loop through the game board and change value of game board square when a player makes a move compare to being implemented by a 2D array. Game board implemented by using an Array instead of a linked list because Array supports Random Access, that indicate elements can be accessed directly using their index. For example, when a player decides to position a cross piece to square 7, square[7] can easily change from "7" to "X" whereas Linked list supports Sequential access, that indicates accessing any square of the game board, we have to traverse the complete linked list to that square sequentially. Therefore, to access the nth element of a linked list, time complexity is O(n) whereas accessing elements within the array is fast with a constant time complexity of O(1). Game board implemented by an array instead of a stack because, in an array, the objects are arranged in a way so it can be accessed at any time randomly whereas, in a stack, the objects arranged in a way where it is only inserted or deleted from one end Insertion and deletion take place within any position. Therefore, accessing a specific square of the game board is more efficient and quick when the game board implemented in an Array instead of a Stack as the time complexity of accessing the value of an array is O(1) and time complexity of accessing the value of stack is O(n). A tutorial from "CProgrammingNotes"[1] have guided me while implementing the game board.

Four stacks are used to implement the undo and redo function. A move is undone every time user types in "10" and a player can redo a move by typing "11". The stacks made are a stack that contains all of the choices which are called "undoChoices", stack that contains all of the marks which are called "undoMarks", stack that contains all of choices that been undone which is called "redoChoices" and stack that contains all of the marks that been undone which is called "redoMarks". Every time a player makes a move, choices, in the "redoChoices" destroyed, marks in the "redoMarks" destroyed, choice pushed into the "undoChoices" and a mark pushed into the "undoMarks". When a player (either player 1 or player 2) undo a move, the mark popped from "undoMarks" and pushed into the "redoMarks", the choice is popped from "undoChoices" and pushed into the "redoChoices". The game then set to the previous game state and is the next player turn to make a decision. Players can keep undoing moves from current state till initial game state where the game board is empty. When a user redoes a move, the un-done move will be re-done where players can redo until all undone moves are on the game board. This operation happens by popping a mark from "redoMoves" and pushing into "undoMoves", popping the choice from "redoChoices" and push into the "redoChoices". When a player makes a move, the players can no longer redo moves that they previously have undone. An article from "GeeksforGeeks"[2] guided me while implementing stacks for these features

Stacks used instead of arrays to implement these features be-

cause a stack is a linear data structure shown by a sequential collection of elements in a fixed order where as an array is a collection of related data values called elements each identified by an index 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 whereas an in an array, it is a random access operation, and everything gets down to start of the array. Insertion and deletion take place in any position. Therefore, faster to undo and redo moves by using a stack instead of an array as is faster to insert values to stacks than inserting values to the array where time complexity of inserting values to the stack is O(1) and time complexity of inserting values to an array is O(n). Stack used instead of a queue to implement the undo and redo features as it is simpler to implement both features using a stack compare to a Queue. The reason is in a stack; the same end is used to input and remove elements whereas, in a Queue, one end used for insertion such as rear end and another end used for deletion of elements such as front end. Dynamic Array implementation of the stack is used instead of a singly linked list implementation of the stack because even though both have average O(1) time when a new move added to the linked list, new allocation needed that can be high contrast to other operations. Arrays do not suffer from this problem.

The "undoChoices" and "undoMarks" stacks used for the redo and undo features is used to replay the game moves after a player wins the game or there is a draw. The feature works by looping through the array from the 0th index to the top value of the stack array. For each iteration, a mark inserted into a square of the game board and the user is asked to press any key to show the next move. This procedure repeats until the last state of the game. The user then asked if he/she wants to play the game again by typing in "yes" or any key to not play again. These features are present in the Two Players game, play with the computer(hard) and the player with computer (easy) versions of the game. Stack array instead of a Queue because it is more straightforward to implement and save memory space as the array implementation of stack used for multiple features. I have used an array implementation of the stack to implement this feature as even though stack have the same time complexity as a linked list, accessing the array of the stack by looping through it is quicker than accessing linked list by looping through linked list. The reason is that time complexity of accessing array is O(1) and time complexity of accessing a linked list is O(n). An article from "GeeksforGeeks" guided me to implement a Linked list.

In order to replay moves of a specific previous game, a user must type in the game id of that specific game. The moves of that game replayed from initial game state to final game state which happens by looping through moves of the game from the first index to the 9th index of the moves array. If the game not found, an error message printed on the console stating "Game Id not found". Feature implemented by writing an array of the "undoMoves" stack array to a file called "list.txt". The function "ReadFromFileToLinkedList()" that has a variable called "count" initialised to 0. The function reads each line from "list.txt" and each node marks array of the linked list is set to the first string of a line and each node choices array of the linked list is set to the second string of a line and each node choices array of the linked list. Each node game id is set to set to count value where the count would iterate at every line. A linear search algorithm would iteratively look for the user inputted game id in the linked list. When the game id located, the game would subsequently replay all of the moves from initial game state to final game state. When the linear search algorithm iteratively looked through the entire game ids of the linked list and have not found any game id that is equal to the user input value, an error message shown. A article from "GeeksforGeeks"[3] have guided me while implementing a linked list for this feature.

The linked list used to store games instead of an array as in an array, elements are stored in a contiguous memory location or sequential manner within memory whereas, in a linked list, new elements can be stored anywhere within memory where the address of memory location allocated to the new element stored within the previous node of linked list. In the array, Insertion operation takes more time as memory location is consecutive and fixed whereas in a linked list, a new element stored at the first free and available memory location with the only single overhead step of storing the address of memory location within the previous node of linked list. Therefore, inserting each game moves from file to linked list takes less time than inserting each move from file to an array where time complexity for insertion to the Linked list is O(1) whereas the time complexity for insertion to the array is O(n). Also, the size of the array must be indicated at the time of array declaration whereas the size of a linked list varies where it grows at run time while more nodes added to it. Therefore, an unlimited amount of games data can be added to the linked list without needing to alter the size of the Linked List. Even though searching a game from hash table time complexity (O(1) on average) is better than searching a game from a linked list (O(n) on average), implementing a linked list is less complicated than implementing a hash table. Also, an unlimited amount of games can be inserted to a linked list without needing to expand the linked list which is not the case with a hash table. Also, a hash function would assign each key to a different bucket, but it is possible that two keys would make an identical hash causing both keys to point to the same bucket that causes collisions which is not the case with a linked list. Therefore, the linked list used to store each game moves instead of a hash table.

Linear search used instead of a binary search to search for the particular game in the linked list as the time complexity(O(n)) searching linked list linearly is same as searching linked list by binary search as both need to look through each linked list node as each node is linked to the node before. The linked list is searched in a iterative way instead of recursively as recursion algorithm has higher space complexity than iterative algorithm.

This feature is similar to the feature where a user can search game moves replays that only he or she played, but the difference is the filename that the moves of each game stored is the name of the user username. This feature allows each user to look at its past moves and to learn how they can improve in later games. The singly linked list used instead of a doubly linked list because singly linked lists have only one pointer whereas doubly linked lists have two field pointers so singly linked lists occupy less memory than doubly linked lists.

For the (Hard)Play With Computer version of the Tic Tac Toe game, I have used a minimax algorithm to allow the computer to consider all possible moves and choose the optimal move every time the computer is making a move on the board while playing against a human. The findBestMove() evaluates all the possible moves using minimax() and then return the best move the maximiser can make. The minimax() algorithm would check whether or not the current move is more optimal than the best move by considering all possible methodologies the game can go and returns the best value for that move, assuming the human also plays optimally. The isMovesLeft() function check whether the game is over and to make sure there are no moves left by returning true or false. Every time is the computer turn; the computer would place the most optimal move on the board. This way is tough for human players to win against the machine. The human player can undo and redo a move, and when the game finished, the game moves subsequently replayed. A article from "GeeksforGeeks"[4] have guided me while implementing a minimax for this feature.

For the (Easy)Player With Computer version of the Tic Tac Toe game, the computer randomly chooses what moves to make by choosing a range of number from 1 to 9 that is not already chosen by the human player. Every time the computer makes a move, index of the game board array is set to "O", depending on what square the computer has chosen such as if the computer chose square 3, square[3] is set to "O". This way, the computer can quickly choose a move and occupies less memory. This feature allows players who are not experienced in the Tic Tac Toe game to have the opportunity to improve their game abilities.

A user can register an account by pressing key "2". The register feature allows users to make an account by inputting username and password. The system would check whether or whether not that the username is already exist in the system by reading through the file "accounts.txt" and compare the first string of each line to the entered username. If the username already exists, the user returned to the menu but if that is not the case, the message printed on console stating "You have successfully, created an account". The password encrypted by using a hashing function, and then both username and password are appended to the "accounts.txt" file in a way where the line has space between username and password. The encrypt() method encrypts the password by subtracting hex value from it. The password encrypted for security reasons such as preventing user accounts being hacked.

A user can log in into their account by pressing key "1". The user can then type into their username and password. The program reads each line of "accounts.txt" file and compares the entered username and password with a username and decrypted password from the file. When the inputted username and password match username and password from a line of the file, the user is signed in. When the inputted username and password is not equal to any username and password from the file, an error message is then displayed, and the user returned to the previous menu. When the user signed in, the user can have the choice to quit, play the multiplayer game, play against the computer(Easy), playing against the computer(hard), search replays(All Games) or search replays(User Games). The quit option will sign user account off and go to the initial menu which happens when user types in "0". The user can completely leave the game by typing in "0" again. A article from "c-program-example"[5] have guided me while implementing the crypt and decrypt functions.

3 Potential Enchancement

In the future, the game board enhanced in a way that players can decide what game board size they want instead of just playing on the 3x3 game board. Before the game starts, the user asked what size of the game board they desire to play on. For example, if the user typed in 6. A 6 x 6 game board would place six rows and six columns. This would enhance the game as more squares the game board has, more challenging for Tic Tac Toe players which would encourage players to play the Tic Tac Toe games multiple times without being bored.

The game would be enhanced in the future by implementing a Hall of fame feature which contains the number of games that each player have won and lost. The number of games a player has won, the higher the player would be placed on the hall of fame. The hall of fame would be in a table format where the columns are the player names; Games played, games won, games lost and player levels. Every time a player wins 50 games, the player level would increase by one. This feature would enhance the game because It would encourage players to play the Tic Tac Toe game multiple times as they would compete with other players to be the best players as possible.

A feature implemented in the future would be a player profile which would also show in a table format that shows player name, player level, games won, games lost, message option and compete for the option. Players can also post photos and update status. A player profile can be searched by going to the search page and typing their name or clicking on their name on the users list section. This feature would enhance the Tic Tac Toe game as each player would have the opportunity to express themselves uniquely to other players through their profile.

A message and compete feature implemented in the future would allow users to message and compete with players around the world. These features can be possible by publishing this game on a cloud server where players must have an internet connection to play against or message players around the globe. These features would enhance the Tic Tac Toe game as it would allow Tic Tac Toe enthusiasts to meet players around the globe that have similar interest as them and to try to outplay as many players as possible which is more enjoyable than two players competing against each other using the same PC.

4 Critical Evaluation

The Two players game has worked extraordinarily well because two players can compete against each other in a way that they take turn making a move where one player places a cross, and another player places a circle on the game board

until a player wins or both draw while the Game board implemented in a 1D array. For example, the game board was first implemented using a 2D array where each player had to state the row and column of which square to place the mark. Therefore, 2D implementation of game board was more time consuming and also took more time for the move to placed on the board compare to the 1D implementation of Game Board which mean 1D implementation of the game board is more optimal than a 2D implementation of the game board.

The undo feature has worked very well because players can successfully undo moves by pressing the key "10" implemented by four stacks called "undoMoves","undoChoices", "redoMoves" and "redoChoices". Undo feature implemented by stacks is the most optimal implementation because when the undo feature first implemented with a 1D array alone, the iterator needs to iterate through to the last index of the array where value removed, and all values of the array must move up the index every time a player undo a move. However, the stack implementation of the undo feature work in a way where every time a player undo a move, the top choice just need to be popped off from "undoChoices" stack and push into the top of "redoChoices" stack, and the top mark just need to be popped off from "undoMarks" stack and push into the "redoMarks". Therefore, insertion and deletion took less time with a stack implementation of the undo feature compare to an array implementation of the undo feature.

The redo feature has superbly worked well because players can successfully redo undone moves by pressing the key "11" implemented by four stacks called "undoMoves","undoChoices", "redoMoves" and "redoChoices". Redo feature implemented by stacks is the most optimal implementation because when the redo feature first implemented with a 1D array alone, the iterator needs to iterate through to the last index of the array where value removed, and all values of the array must move up the index every time a player redo a move. However, the stack implementation of the redo feature work in a way where every time a player redo a move, the top choice just need to be popped off from "redoChoices" stack and push into the top of "undoChoices" stack and the top mark just need to be popped off from "redoMarks" and pushed into the top of "undoMarks". Therefore, insertion and deletion took less time with a stack implementation of the redo feature compare to an array implementation of the redo feature.

The replay feature has worked excellently well because It has successfully sequentially replay each move from initial game state to final game state every time the player presses a key quickly and efficiently while looping through an array implementation of the stack. Replaying the moves from initial game state to final game by looping through an array compare to looping through a linked list as accessing elements within an array is fast with a constant time complexity of O(1) whereas to access an element of a linked list, the time complexity is O(n).

Searching replays for all games and games for the login player have worked partially well. Users can efficiently search up moves on a particular previous game in an efficient manner by typing in the game id of a specific game, and the moves of the game are replayed sequentially from initial game state to final game state. However, searching through the Linked List time complexity is O(N) which worse than searching through a Hash table which is O(1) as all nodes are connected. However, using a Linked List is still optimal as an unlimited amount of game moves ability to be added to the Linked List which is not the case with Hash table.

The hard version of the player against computer Tic Tac Toe game has worked well as nearly impossible to win against the computer because computer using the minimax algorithm that recursively looks at all of the possible moves and places the best move on the board when its turn in a very efficient and fast manner. Minimax algorithm is the most optimal algorithm to implement this game as when I tested this game by multiple times by playing against the computer; I have not managed to beat the computer, I either lose to the computer or draw against the computer. Also, whenever it is the computer turn, the computer only took a few microseconds to place a move on the board.

The easy version of a player against computer has amazingly worked well as players cannot cheat and computer only randomly places the piece on an empty square expeditiously and efficiently by using the "rand() mod 9" which was most optimal implementation. The proof is that when the feature initially implemented in a way that the program chooses random numbers from an array of numbers from 1 to 9 in order to place a piece on the board, It took incredibly a very long time to do so compared to using "rand() mod 9" algorithm.

The registered feature has incredibly worked well as the user can only type in a username that does not already exist in the system and also the password is encrypted before it is written to the file, so the player does not have to worry that their account hacked. When username already exists, an error message displayed on the console, and the user returned to the previous menu. When a player successfully makes an account, a message stating that displayed on the console and the player returned to the previous menu.

The login feature works well as it successfully verifies entered username and password and allows the user to go to the next menu. When the username and password are incorrect, an error message displayed on the console, and the user returned to the previous menu.

5 Personal Evaluation

This coursework has taught me many things. The coursework has taught me the importance of memory management on the performance of my application which would improve my future projects as I am now able to effectively choose what data structure is most suitable for making application to use less memory space. The coursework has taught me various decision-making skills such as what algorithm should I use on a particular data structure for a specific function or feature based on time and space complexity which would improve my future projects. The reason is that I am now able to effectively choose what data structure and algorithms to use in order to make sure the features of my future applications to run efficiently and quickly without using up much memory space. The coursework has also taught me how to debug

syntax or run time errors more quickly by using printf() function and breakpoints which would improve my future projects as I will able to fix errors more quickly so would manage time more efficiently and use more time on implementing other features of the future applications.

A challenge I have faced was replaying moves as the array that was meant to hold all of the moves that placed on the game board after the game finish was pointing to a memory slot that does not exist. The methodology that I have used to solve this challenge was that I had used the printf function in the for loop that been used to loop through the choices and marks of the game to subsequently replay the game moves so I can see the moves and choices printed out on screen. I have noticed that it was printing out all the seven moves and choices of the game but printing out strange symbols after this. The reason was that only seven moves had been made in that game while the for loop while making the program to loop through the array till iterator value was 9. I have decided to modify the for loop parameters making the for loop to loop through the "undoChoices" from index 0 to top value of "undoChoices" stack. The result of this was successfully managing to sequentially replay moves from initial game state to the final game on the game board.

A challenge faced while implementing this game was that the program kept crashing when the user goes to the Games Replays section. The methodology that I have taken to overcome this challenge is by making a function for test purposes that would read and print out all of the games marks and choices on the console. When it has successfully done that without making the program to crash, I have used the printf() function within readingFromFileToLinkedList() function to print out choices and marks of each node after it read from the file to a node of the linked list. Only one node marks and choices been printed out on the console and then the program crashed which made me realised that there is a problem with how data added to a node of linked list. After a quick google search, I have found out that I have not been setting next node pointer to a new game after choices and moves of a node of the linked list within the addGame() function. I have set the next pointer of the linked list to the new game node. The result of this was that users can now go to the Games replay section without the program crashing and enter a game id to see the moves replay of a specific game.

The final challenge faced while implementing this game was in the first test of minimax algorithm used to make computer to place the most optimal move of all possible moves on the board where computer piece was "O". The challenge was that when human first makes a move, the whole board start to be filled with Xs and then player one wins. The methodology that been taken to try overcome this challenge is that within the findBestMove() function, I have used printf to print out the score while analysing all possible moves. The program was rerun, and the game was printing out "X" multiple times. I have then analysed the minimax(), findBestMove() and evaluate() function and compared it with the minimax algorithm pseudo-code, this has made me realise that there are many mistakes in these functions, so I have made many modifications and reran the program to check the scores values. The value printed out was 0 a few times then -10 printed out which shows that the computer managed to find out what is the most optimal move. The result was that the when player one makes a move, the computer looks at all possible moves and chooses the best move which repeats until computer or human wins the game, or there is a draw.

Overall, I think I have done amazingly well implementing this Tic Tac Toe game as I have managed to make a turn-based Tic Tac Toe game where two players can compete with each where player one make a move by placing "X" on the board and player two can make a move by placing "O" on the board. Players can also undo and redo moves where undo function can successfully make the game to go to the previous game state, and the redo function can redo all of the undone moves. Players can do all these actions until a player wins or there is a draw. After the game finished, moves are subsequently replayed, and also players can manage to replay moves of any previous games they wish subsequently. All of these features meet the coursework specification. I have also made features that extend the specification which is allowing users to play against an A.I, computer, register account, login into the game and each player can observe subsequently replay of moves of any game that specific player has played before. All of these features implemented in efficient data structures and algorithms.

References

[1] cProgrammingNotes, "Tic tac toe game,"

[2] . GeeksforGeeks, "Stack data structure," 2017.

[3] . GeeksforGeeks, "Linked list — set 1 (introduction)," 2017.

[4] . GeeksforGeeks, "Minimax algorithm in game theory — set 3," 2016.

[5] . c-program example, "C program to encrypt and decrypt a password," Apr. 2012.