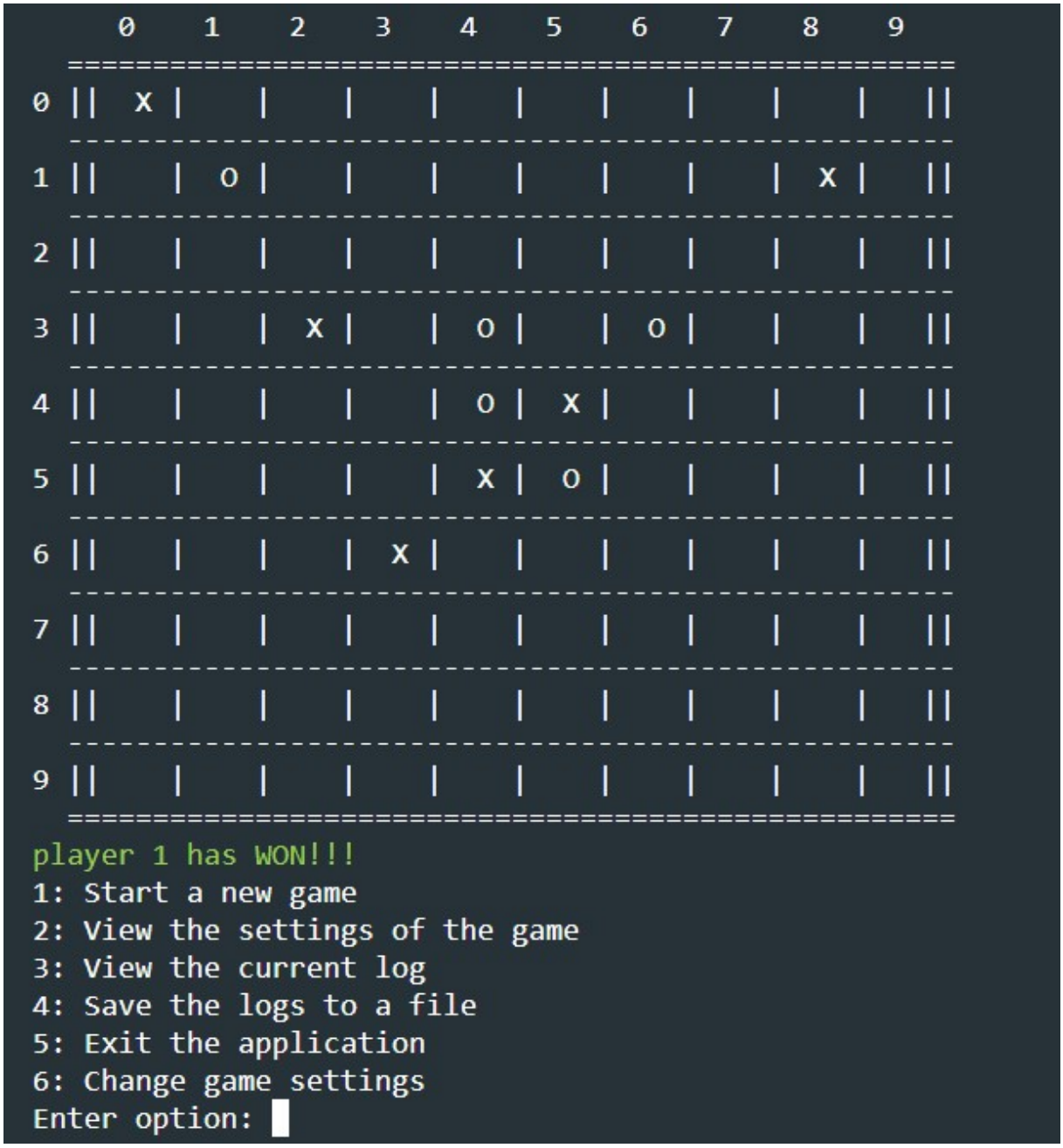
CURTIN UNIVERSITY
DEPARTMENT OF COMPUTING

Tic Tac Toe Report

Student name: *Tawana Kwaramba: 19476700*

Course: *Unix and C Programming (COMP1000)* – Mr: *Mark Upston*
Due date: *October 21st, 2019*



Contents

List of Figures

1. Introduction

This programming report aims to clarify the approach in designing and testing the m-n-k tic tac toe game created in C89. The game is a normal game of tic tac toe except the width, the height and the number of matching tiles can be changed throughout the game.

2. Programme files

2.1. README.md. Documentation to get a quick overview of the functionality, purpose, files, and bugs of the programme discovered through testing of the code but weren't solved because of limited time or lack of experience.

2.2. fileInterface.c. The overall purpose of the file interface is to handle all the file input and output of the programme i.e. reading game settings and writing game log. Furthermore, the file contains other auxiliary function which help in dealing with the file input and output of the programme these include functions such as setInvalid(), processLine(), parseChar(), and isDuplicates().

2.3. gameGraphics.c. The overall purpose of this file is to deal with displaying of game graphics to the terminal i.e. displaying the game-board, displaying the character avatars on the board and reprinting the board with the updated positions. Furthermore, this file also contains auxiliary functions which help the file to achieve this goal.

2.4. gameInterface.c. The overall purpose of this file is to bring together other components of the game such as game logic and game graphics to create a playable game. Therefore, this file is where the game is played, and the game coordinates are sourced from.

2.5. gameLogic.c. Game logic is synonymous with the man behind the curtain in the wizard of oz. Thus, this file is responsible for the back-end of the game hence, it's responsible for moving the player, setting the tiles to the appropriate states, switching the players, validating the user input, determining the winner in each plane and calculating the area of the board (this is helpful when calculating if the game has resulted in a draw).

2.6. LinkList.c. The overall purpose of this file is to contain functions relating to the functionality of the abstract data structure of a doubly linked, double-ended linked list. Therefore, it contains functions which relate to creating the linked list, inserting first, inserting last, removing last, removing first, displaying to the terminal and to a file through the assistance of function pointers, and freeing the linked list.

2.7. logInterface.c. The log interface is to facilitate the functionality of logging game events into a game log which is comprised of the abstract data structure of a linked list. Therefore, the log interface contains functions which are responsible for creating the log, logging the game settings, logging the game number, logging the game-play, displaying the log to the terminal, displaying the log to a file, generating the log name and freeing the created log.

2.8. memoryInterface.c. The memory interface is to facilitate the creation and freeing of two-dimensional arrays. This function didn't fit anyway else hence their own files was made.

2.9. userInterface.c. Is to facilitate the user's interaction with the game hence, it can be thought of as a controller for the programme giving power to the user to choose what they want the programme to do. Therefore, this is where the main menu for the programme is displayed and where other functions are invoked in relation to the user's request. Additionally, the user interface facilities different modes of the programme hence, it facilitates the secret and editor mode of the programme.

3. Implementing logs

The storing of the game events (logs) was done through creating a linked list where a log game structure is stored. The game log structure consisted of four strings which are dedicated to holding information about the game, the information each string in the structure is meant to hold is as follows; the first variable "mssg" is designed to hold the game settings or the game number, the "turn" variable is designed to hold the string "turn:" alongside with the number of turns which has occurred in the game, the "player" variable is designed to hold the string "player:" alongside the players avatar which is either 'X' or 'O', and the "pos" variable is designed to hold the string "Location:" alongside the coordinates of the players turn.

With this structure in mind storing the game events are quite simple. Before we play a game we first insert last the game settings and the game number in the "mssg" field of the structure .Then when the game is in play an insert last will be done with the structures fields of turn, player, and position in relation to the game's events. Given, the case of editor mode when settings can be changed during the programme, an insert last will be done every time the settings are changed through the sixth menu option.

The implementation of printing the logs to the terminal is through a recursive approach. It is evident that the steps of printing out a list node is the same for every list node in the linked list i.e. you get the list node, give the list node to a function pointer which type-casts that list nodes value into the appropriate data and prints it out to the terminal hence, the problem of printing can be broken into smaller steps which is a criterion for recursive implementation. Furthermore, we can observe that we are going to print the linked list until there are no more list nodes on the linked list thus, in other words we're going to print out the list node until it reaches NULL which is a reachable base case which is another criterion needed for a recursive implementation. Hence, printing to the terminal would consist of a wrapper method, the purpose of this wrapper method gives the recursive function a place to start and a place to come back too once it has finished its recursive call of subsequent stack frames. This wrapper method will check if the linked list is empty if it's empty do nothing, if it's not empty the wrapper method will invoke the recursive display function.

The recursive function will check if the node is null, if it's not null use the function pointer to display the value of the node in the same manner as described above, thereafter the algorithm will call its self with the next node hence, creating a stack frame

which has exact copies of previous stack frames apart from the value of the node. The above process will keep repeating until the base case is reached. Then the code will call back to all the created stack frames completing the rest of the code and deleting itself until it returns to the wrapper method.

They were no complications in printing the contents of the logs to the terminal as this problem was synonymous to printing out trees in the unit of Data Structures and Algorithms hence, similar approaches were used, and printing out to a file was the same problem except printing out to the standard output stream.

4. Brief discussion about testing

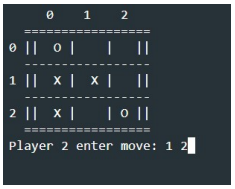
The testing of the programme was done in the following manner; design each function of the programme, test the functions independently with data which represents different cases the function can expect, in parallel Valgrind was used to identify memory issues. Furthermore, some components such as playing the game were required to be tested manually as errors in the game-play can be only effectively discovered in manually playing the game. After, individual testing of functions the user interface and the main was constructed which allowed for further manual testing of the game as a whole through Valgrind and various in-putted data.

5. Demonstration of the game

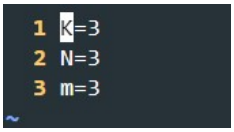
(a) Command line to execute programme

(b) interaction with the programme menu

Figure 1: interaction with menu and command line



(a) Interaction with the game menu



(b) settings of the file

Figure 2: interaction with the board and game settings

```
SETTINGS
    M:3
    N:3
    K:3

GAME 1:
    turn: 1
    Player: X
    Location: 0,1

    turn: 2
    Player: O
    Location: 2,2

    turn: 3
    Player: X
    Location: 1,1

    turn: 4
    Player: O
    Location: 0,0

    turn: 5
    Player: X
    Location: 0,2

    turn: 6
    Player: O
    Location: 1,2

    turn: 7
    Player: X
    Location: 2,1

```

(a) Interaction with the game menu

Figure 3: Contents of the log file