TIC-TAC-TOE PROGRAM REPORT

Unix & C Programming Assignment 2019

Liam Ryan
19769811

Table of Contents

1 Files

1.1 TicTacToe.c

This file contains the main from the program. It checks that a settings file has been included as a parameter and then it attempts to load the settings from this specified settings file. If the settings are valid then the menu is called (inside UserInterface.c) with m, n and k passed in as integers.

I originally had my main inside my UserInteface, but chose to create a wrapper file and then called UserInterface once I knew the game could be run (the settings are valid), as this is much better programming practice.

1.2 UserInterface.c

This file contains the main menu for the program. I contemplated having all user output and input inside this file, but I chose to only include the main menu in this file. I preferred to structure my program such that each file being responsible for a certain aspect of the program, rather than having for example a FileIO file that does all the file input/output and an Input/UI class that does all the terminal interaction. My program has printing, scanning and FileIO in many different files but each section of the program is controlled by a single file entirely, this was a decision which I believe is best. UserInterface takes in the menu input and calls the various functions in the various files to perform specific functions, it also displays a different menu and allows/restricts access to certain pieces of functionality depending on what mode/s (EDITOR/SECRET) the program is in.

1.3 Settings.c

This file is responsible for the FileIO, validation and allocation of the settings and is called by in TicTacToe.c. The bulk of this file is the loadSettings() method which returns true/false whether the settings are valid, if they are valid it exports values for m, n and k. I also chose to validate the value for each dimension to be from 1-26, as I used a letter-number notation for input and output of the board similar to chess which I will discuss further on.

1.4 GameManager.c

Handles everything to do with the creation and running of the game of Tic-Tac-Toe itself. The struct for a game of Tic-Tac-Toe (Game) is defined, created and freed here. Included in this file are methods that display the game, check if the game is finished and the placing of a token. This file also adds to the log as each successful move is placed.

1.5 Input.c

This file contains a various assortment of methods which aid the program:

- inputInt(): allows for validated input of an int between min and max
- inputChar(): allows for validated input of a char between min and max
- toUppercase(): converts a char to uppercase if not already

As I had input and output in many different places in my program it was helpful to create an independent file with these useful functions which I could use throughout the program.

1.6 Logging.c

This file is responsible for all functionality to do with logging: creation, adding to log, displaying, printing to file and freeing. There is FileIO in this class with the saveLog() function and also user output with the viewLog() function. The log itself is a LinkedList (generic) and I used the generic linked list created in Prac 7 for the TicTacToe program.

This file also has the declaration and creation of the Move struct, which represents a single move in a game of Tic-Tac-Toe (contains: int turn, char player, int colIdx, int rowIdx).

1.7 UCPLinkedList.c

This file contains a generic linked list as was created in Practical 7. Performs, creation, addition, removal, freeing and printing of the list.

2 Logs

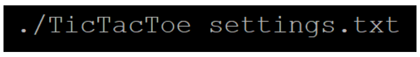
There is only one log per instance of the program and it is created, using createLinkedList(), at the very start of the UserInterface – once we know that the settings are valid and the program can be run. I pass a pointer to the log through the various functions into runGame() and subsequently the other functions it calls inside GameManager.c so that I can add a move to the log every time a valid move has been played. I would create a Move struct after each valid move and then use insertLast() to add this to the log.

I was unsure of how to indicate that a game has finished, the problem was that when we are displaying/writing the log how do we know when a game has ended so we can indicate the start of a new game? I contemplated ideas such as a LinkedList of logs (being LinkedLists themselves) and have each game having its own log. I also contemplated indicating a game was finished by having some non-Move struct as the data in the node. The solution I chose was to have one continuous linked list of Move structs for consistency and that I would indicate that a game has finished by having a Move with a negative turn/total. Turn is initialised to 1 every time and increments with each move. I set turn to -1 if Player 1 won, -2 if Player 2 won and -3 if it was a draw. I check for the Move struct in each node having a negative 'turn' to know when a game has finished. All three possible values are negative, however I chose to differ between the different values based on how the game ended so that the program could in future be edited easily so that it also creates a statement of how the game finished.

3 Demonstration

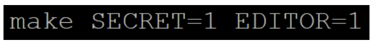
3.1 Command-line

The program can be run by using ./TicTacToe <settingsFile>. An example of this is shown on the right (Fig. 1). The program can be created utilising the makefile by entering "make". The program can also be created in different modes by including SECRET=1 and/or EDITOR=1 after the make command (Fig. 2).



```
./TicTacToe settings.txt
```

Fig. 1: Command-line execution

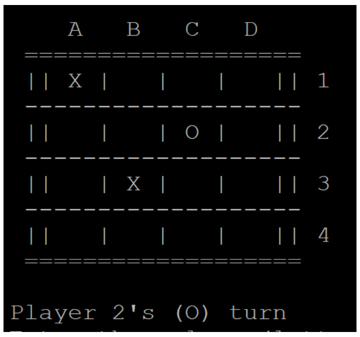


```
make SECRET=1 EDITOR=1
```

Fig. 2: Making in different modes

3.2 Output of the board

The board is output such as is shown on the right (Fig. 3). I chose to have the columns being represented by letters and rows being represent by numbers as is commonplace in many board games including chess. I found this much better than a letter-letter or number-number system as the user can identify a cell much easier and without difficulty. The program also states whose turn it is on each move as shown.

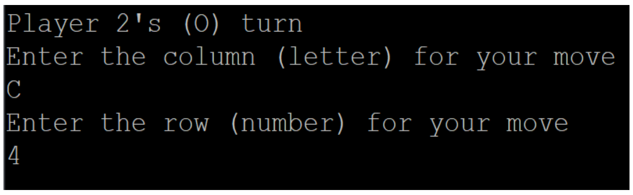


```
    A   B   C   D
  ===================
  || X |   |   |   || 1
  -------------------
  ||   |   | O |   || 2
  -------------------
  ||   | X |   |   || 3
  -------------------
  ||   |   |   |   || 4
  ===================


Player 2's (O) turn
```

Fig. 3: Board display

3.3 User input to board

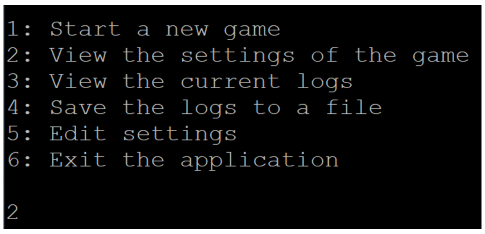
The user input to the board used this letter-number notation as commonly used in board games such as chess. The user is asked for and then enters in the column (letter) and then is asked for and then enters in the row (number) for their move (Fig. 4). The input is also non case sensitive for the letter input. Appropriate error checking is done, repetition and error message are included until a valid input.



```
Player 2's (O) turn
Enter the column (letter) for your move
C
Enter the row (number) for your move
4
```

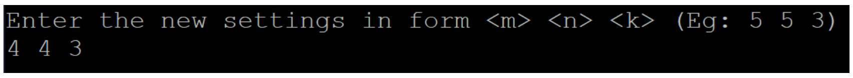
Fig. 4: User input

3.4 Other user input



```
1: Start a new game
2: View the settings of the game
3: View the current logs
4: Save the logs to a file
5: Edit settings
6: Exit the application

2
```

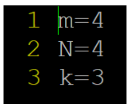
Fig. 5: Menu selection (Editor mode)



```
Enter the new settings in form <m> <n> <k> (Eg: 5 5 3)
4 4 3
```

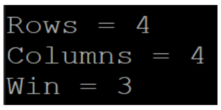
Fig. 6: Changing of settings (Editor mode only)

3.5 Input file/ Display of current settings



```
m=4
N=4
k=3
```

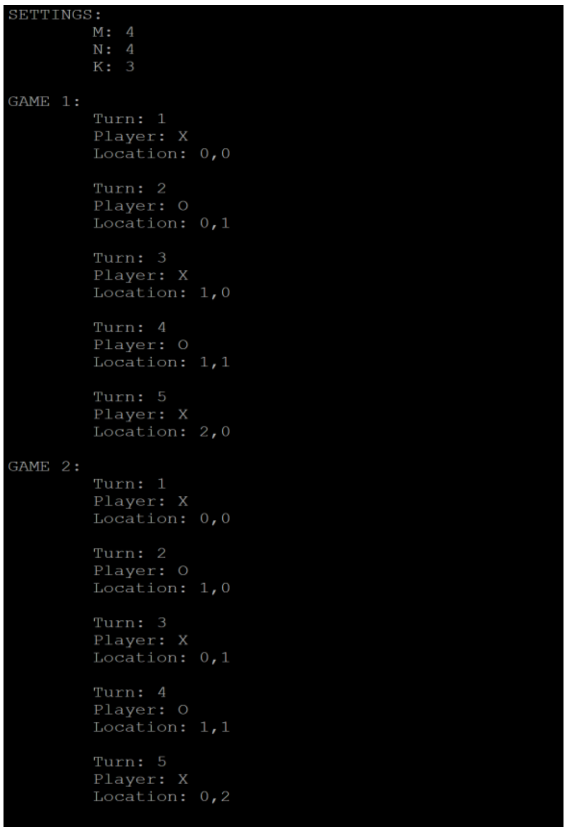
Fig. 7: Content of settings/input file



```
Rows = 4
Columns = 4
Win = 3
```

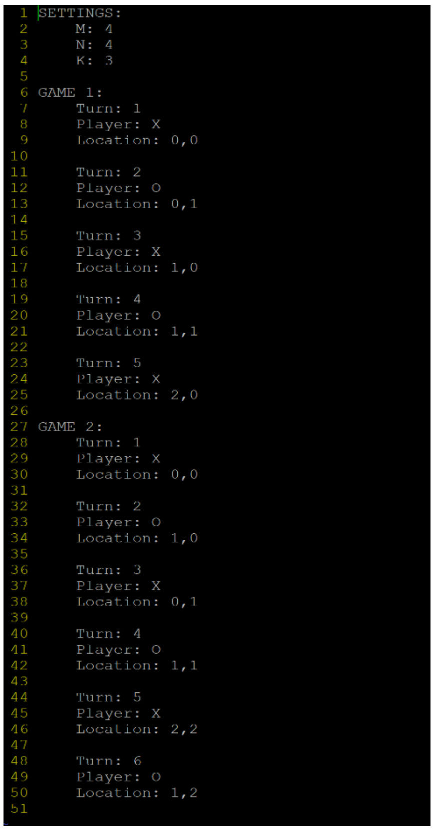
Fig. 8: Display of settings

3.5 Display of Logs/Contents of Log File



```
SETTINGS:
        M: 4
        N: 4
        K: 3

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

        Turn: 2
        Player: O
        Location: 0,1

        Turn: 3
        Player: X
        Location: 1,0

        Turn: 4
        Player: O
        Location: 1,1

        Turn: 5
        Player: X
        Location: 2,0

GAME 2:
        Turn: 1
        Player: X
        Location: 0,0

        Turn: 2
        Player: O
        Location: 1,0

        Turn: 3
        Player: X
        Location: 0,1

        Turn: 4
        Player: O
        Location: 1,1

        Turn: 5
        Player: X
        Location: 0,2
```

Fig. 9: Display from viewLog()



```
SETTINGS:
    M: 4
    N: 4
    K: 3

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

    Turn: 2
    Player: O
    Location: 0,1

    Turn: 3
    Player: X
    Location: 1,0

    Turn: 4
    Player: O
    Location: 1,1

    Turn: 5
    Player: X
    Location: 2,0

GAME 2:
    Turn: 1
    Player: X
    Location: 0,0

    Turn: 2
    Player: O
    Location: 1,0

    Turn: 3
    Player: X
    Location: 0,1

    Turn: 4
    Player: O
    Location: 1,1

    Turn: 5
    Player: X
    Location: 2,2

    Turn: 6
    Player: O
    Location: 1,2

```

Fig. 10: Contents of log file

External References

- Techie Delight. (2019). *Print current date and time in C - Techie Delight*. [online] Available at: https://www.techiedelight.com/print-current-date-and-time-in-c/ [Accessed 20 Oct. 2019].

Self References

I used modified and direct pieces of code that I had already created in the practicals of UCP. These include:

- LinkedList.c : a generic linked list (Prac 7)
- toUppercase() : original version converted a string to uppercase; I modified this and added extra functionality to convert a single character to uppercase with validation (Prac 4)