

Unix and C Programming

Final Assignment Report

Author: Anton Rieutskyi

Student ID: 19587673

Lecturer: Antoni Liang

Unit Coordinator: Mark Upston

Date of Submission: 27/10/2019

Contents

1. Introduction	1
2. Discussion of Project Files	1
2.1. game.c/game.h	1
2.2. interface.c/interface.h	1
2.3. fileIO.c/fileIO.h	1
2.4. linkedList.c/linkedList.h	2
2.5. bool.h	2
2.6. main.c	2
3. Logging Functionality	2
4. Gameplay Examples	2
5. References	3
Appendix 1. Figures	4

1. Introduction

The purpose of the assignment was to design and implement (in C89) an M-N-K tic-tac-toe game, where M is the width of the board, N is the height and K is the number of tiles in a row required to win (Curtin University 2019). Per the Assignment specification, the program was expected to be able to: provide the user with a terminal based menu; read game settings from a text file; edit the settings based on user input; play out a full game of tic-tac-toe; log all players' moves; and save the logs to a file. This report outlines the program's design by discussing the main project files and the author's approach to recording and printing the game logs, and provides some examples of the game's functionality.

2. Discussion of Project Files

2.1. `game.c/game.h`

The `game.h` file defines several structs that are used to store gameplay information, such as game settings, game board state, coordinates, turn logs and game logs. It also defines two enumerations that are used in gameplay, `TicTacTile` and `GameState`. `TicTacTile` is an enumeration used for representing the current player, as well as their respective tile. It consists of an empty tile (`TicTacNone`), the two player tiles (`TicTacX` and `TicTacO`), and a 'terminating' tile (`TicTacTotal`) that represents the total number of tile types (including the empty tile). This design enables potential expansion in the future, allowing for an arbitrary number of players as long as they are added before the terminating tile. The `GameState` type is an enumeration of the possible game states, that changes based on pre-processor variables to enable conditional compilation. This type is used to determine which interface to display to the user.

`game.c` defines all of the gameplay functions required to play the game. These include the main game loop, which continually displays the main menu and updates the game state based on user input until the user chooses to exit, the main menu functions which are called based on the menu choice, a set of validating functions that check every game move, and a set of functions that are used to manipulate the gameplay structs: `gameboard`, `game log` and `turn log`.

2.2. `interface.c/interface.h`

`interface.h` defines the graphical constants needed for displaying the game's interface. These include the ANSI escape sequences for changing terminal colour output (ANSI Escape Sequences n.d.), clearing the terminal screen (The Linux Documentation Project n.d.), and switching to the DEC special graphics character set to access the box-drawing characters (Williams n.d.). The `interface.c` file enables interacting with the user through the terminal console. It defines the functions for prompting the user for single integers and pairs of coordinates, and functions for outputting all of the game's information to the user, like drawing the main menu and the game board. It also implements some static functions intended for use only inside of `interface.c`, which change the terminal's colours and character sets for drawing purposes. Lastly, `interface.c` defines the `printErr` function that is used for outputting highlighted error messages to the user.

2.3. `fileIO.c/fileIO.h`

These files are intended for reading game settings from a text file and creating log files for writing. They define two constants that set the maximum length of input lines and output file names, a function for reading game settings from a file, a function for validating the game settings, and a function for creating log files based on the current date, time and game settings. When reading from a file, the program validates the data and prints an error if the input file's format is wrong. The save file functionality is achieved using the `time.h` library to get the current time and convert it to a readable `struct tm` (TutorialsPoint n.d.). The data from the struct is then used to construct a file name using the current hour, minute, day and month, as well as the game parameters. The string is used to open a new output file which is passed down to the calling function.

2.4. `linkedList.c/linkedList.h`

The purpose of the `linkedList` files is to provide generic linked list functionality to the game. `linkedList.h` defines the two structs used for linked lists – `LinkedList` and `ListNode`. The linked lists are implemented doubly linked and double-ended for easy insertion and deletion, and can store pointers to any data types in the form of a void pointer. The file also defines two function pointer types – a `PrintFunc` and a `FreeFunc`, which are used in printing and freeing the lists. The functions that manipulate linked lists are defined in `linkedList.c`, though only the functions needed for the purposes of the game were implemented, which are: `create list`, `insert last`, `print element`, `print list` and `free list`. All code in the `linkedList` files is derived from Rieutskyi A., 2019.

2.5. `bool.h`

The C89 standard does not define the Boolean data type, so `bool.h` was implemented to make logical expressions more intuitive. It defines a new `BOOL` data type and its two constants, `TRUE` and `FALSE`. `BOOL` is simply a typedef of the `int` datatype, `FALSE` is defined to be 0, and `TRUE` is defined to be `!FALSE` (not false). This implementation allows any non-zero value to be considered true, which is less restrictive than defining `TRUE` to be 1 and is more in line with how C treats its logical expressions.

2.6. `main.c`

This file does not introduce any new functionality and is only used as a starting point for the game. The `main` function ensures that the user has inputted the correct number of command-line arguments, then tries to read the settings file and start the game. If the settings file could not be read properly, the program displays an error and exits, and if the settings were read successfully, the program enters the main game loop until the user exits.

3. Logging Functionality

The game logs all of the players' turns using two kinds of structs (not including the linked lists): a `GameLog` struct and a `TurnLog` struct. `GameLog` consists of a `Settings` struct (but only if `Editor` is defined) and a pointer to a linked list of turn logs. A `TurnLog` consists of the turn number, the player that made the turn (as a `TicTacTile` variable), and the position of the tile they placed (as a `Coordinates` struct). The `GameLog` structs are stored in another linked list during the game, making the complete log a 2-dimensional linked list (linked list of linked lists). A new `GameLog` is created at the start of each game and new `TurnLogs` are added to it each turn. At the end of the game, the game log is added to the log list

Printing of the game logs is achieved using the `PrintFunc` type defined in `linkedList.h`. `PrintFunc` is a pointer to a function that imports a generic pointer, as well as a pointer to an output stream, which enables it to conveniently print out the contents of a linked list to either the terminal console (`stdout`) or to an output file. One of the main complications while designing the printing functionality was the numbering of the items, since both the games and the turns had to be numbered. There were two main options to achieve this: have the structs save their number themselves, or compute their number based on their position in the list. Both solutions were implemented, since the turn logs keep track of the turn numbers while the game logs do not. For this to work, two functions had to be implemented: `printList` and `printElement`, which print either the entire list or a specific element respectively.

4. Gameplay Examples

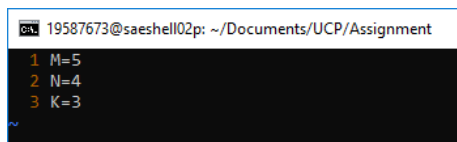
The game can be run using the `TicTacToe` executable with one command-line parameter that represents the settings file (as seen in *Figure 2*). The settings file must have the same format as in *Figure 1*, though the three settings may be present in any order and in upper or lower case. If an invalid setting is found, the program will display an error message and exit. When the game starts, it displays a welcome message to the user and prompts them to press enter when they are ready (*Figure 3*). A menu is then

displayed, where the user can choose to start a new game, view/edit settings, view/save logs or exit the game (*Figure 4*). If the user chooses new game, the terminal screen gets cleared and the game board is drawn using box-drawing characters (*Figure 5*). Each player is then prompted to make their turn on by one until one of the wins (*Figure 6*). The game then returns to the main menu screen, where the logs of the previous game and the games before it can be viewed and saved. The format for viewing the logs on the terminal screen and saving into a file is the same, as can be seen in Figures 8 and 9. Throughout the whole game, the program validates the user's input, making sure they do not enter any unexpected values (*Figure 7*).

5. References

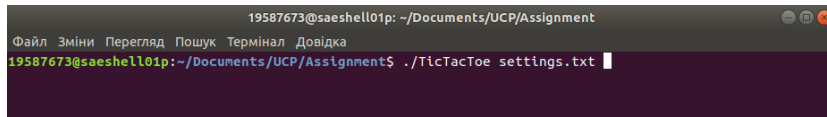
- n.d. "ANSI Escape Sequences." *ascii-table.com*. Accessed October 27, 2019. <http://ascii-table.com/ansi-escape-sequences.php>.
- Curtin University. 2019. "Unix and C Programming Semester 2 2019 Assignment Specification." Accessed October 27, 2019. https://lms.curtin.edu.au/bbcswebdav/pid-7207720-dt-content-rid-37531680_1/courses/2019_2_COMP1000_V1_L1_A1_INT_670711/UCP_2019S2_Assignment.pdf.
- Rieutskyi, Anton. 2019. "UCP Practical 7." Curtin University.
- The Linux Documentation Project. n.d. "Bash Prompt HOWTO: 6.2. Cursor Movement." *tldp.org*. Accessed October 27, 2019. <https://www.tldp.org/HOWTO/Bash-Prompt-HOWTO/x361.html>.
- TutorialsPoint. n.d. "C library function - localtime()." *tutorialspoint.com*. Accessed October 27, 2019. https://www.tutorialspoint.com/c_standard_library/c_function_localtime.htm.
- Williams, Paul Flo. n.d. "Digital VT100 User Guide: Table 3-9 Special Graphics Characters." *vt100.net*. Accessed October 27, 2019. <https://vt100.net/docs/vt100-ug/table3-9.html>.

Appendix 1. Figures



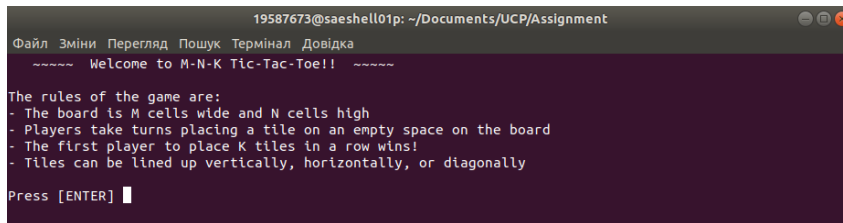
```
19587673@saeshell02p: ~/Documents/UCP/Assignment
1 M=5
2 N=4
3 K=3
```

Figure 1: A sample input file



```
19587673@saeshell01p: ~/Documents/UCP/Assignment
Файл Зміни Перегляд Пошук Термінал Довідка
19587673@saeshell01p:~/Documents/UCP/Assignment$ ./TicTacToe settings.txt
```

Figure 2: The command used to execute the game



```
19587673@saeshell01p: ~/Documents/UCP/Assignment
Файл Зміни Перегляд Пошук Термінал Довідка
Welcome to M-N-K Tic-Tac-Toe!!

The rules of the game are:
- The board is M cells wide and N cells high
- Players take turns placing a tile on an empty space on the board
- The first player to place K tiles in a row wins!
- Tiles can be lined up vertically, horizontally, or diagonally

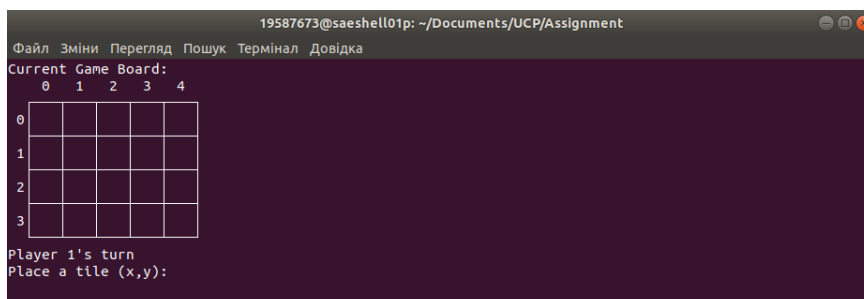
Press [ENTER]
```

Figure 3: The Welcome message at the start of the game



```
MAIN MENU:
>> 1. New Game
>> 2. View Settings
>> 3. Edit Settings
>> 4. View Game Log
>> 5. Save Game Log
>> 6. Exit
Please select an option:
```

Figure 4: The main game menu



```
19587673@saeshell01p: ~/Documents/UCP/Assignment
Файл Зміни Перегляд Пошук Термінал Довідка
Current Game Board:
  0 1 2 3 4
0
1
2
3

Player 1's turn
Place a tile (x,y):
```

Figure 5: An empty game board

```
19587673@saeshell01p: ~/Documents/UCP/Assignment
Файл  Зміни  Перегляд  Пошук  Термінал  Довідка
Current Game Board:
  0   1   2   3   4
0  [ ] [0] [ ] [ ] [ ]
1  [ ] [X] [0] [ ] [ ]
2  [ ] [X] [ ] [0] [ ]
3  [ ] [ ] [ ] [X] [ ]
Player 2 has won!
Press [ENTER]
```

Figure 6: The game board after a player has won

```
19587673@saeshell01p: ~/Documents/UCP/Assignment
Файл  Зміни  Перегляд  Пошук  Термінал  Довідка
Current Game Board:
  0   1   2   3   4
0  [ ] [ ] [ ] [ ] [ ]
1  [ ] [X] [ ] [ ] [ ]
2  [ ] [ ] [ ] [ ] [ ]
3  [ ] [ ] [ ] [ ] [ ]
Player 2's turn
Place a tile (x,y): 4,4
ERROR: Coordinates outside of valid range
Place a tile (x,y):
```

Figure 7: Game's response to invalid input

```
19587673@saeshell01p: ~/Documents/UCP/Assignment
Файл  Зміни  Перегляд  Пошук  Термінал  Довідка
Please select an option: 2
The game's settings are:
  Board size: 5x4
  Win condition: 3 tiles in a row
MAIN MENU:
>> 1. New Game
>> 2. View Settings
>> 3. Edit Settings
>> 4. View Game Log
>> 5. Save Game Log
>> 6. Exit
Please select an option: 4
#####
###  GAME  1  ###
#####
SETTINGS:
M: 5
N: 4
K: 3
Turn: 1
Player: X
Location: 0,0
Turn: 2
Player: 0
Location: 1,1
Turn: 3
Player: X
Location: 0,1
Turn: 4
Player: 0
Location: 2,2
Turn: 5
Player: X
Location: 0,2
MAIN MENU:
>> 1. New Game
>> 2. View Settings
>> 3. Edit Settings
>> 4. View Game Log
>> 5. Save Game Log
>> 6. Exit
Please select an option:
```

Figure 8: Game's log displayed in a terminal

```
19587673@saeshell01p: ~/Documents/UCP/Assignment
Файл  Зміни  Перегляд  Пошук  Термінал  Довідка
1 #####
2 ###  GAME  1  ###
3 #####
4 SETTINGS:
5 M: 5
6 N: 4
7 K: 3
8
9 Turn: 1
10 Player: X
11 Location: 1,1
12
13 Turn: 2
14 Player: O
15 Location: 4,3
16
17 Turn: 3
18 Player: X
19 Location: 0,0
20
21 Turn: 4
22 Player: O
23 Location: 1,2
24
25 Turn: 5
26 Player: X
27 Location: 0,1
28
29 Turn: 6
30 Player: O
31 Location: 2,2
32
33 Turn: 7
34 Player: X
35 Location: 0,2
36
37 #####
38 ###  GAME  2  ###
39 #####
40 SETTINGS:
41 M: 7
42 N: 5
43 K: 4
44
45 Turn: 1
46 Player: X
47 Location: 1,4
48
49 Turn: 2
50 Player: O
51 Location: 2,3
52
53 Turn: 3
54 Player: X
```

1,1 Bropi

Figure 9: Sample log file. Colours were added by vim, not actually part of the output