SnakeGame

Introduction

The aim of this project was to study Linux terminal capabilities, make experiments with the its settings and eventually implement a simple interactive game. The program was written in C99 language. Prior development the language features thoroughly practiced to make possible use of advanced language features.

Methods and materials

Program operation

There are three main variables to represent current game state. First one is a structure named ‘snake’ and it describes one snake. It contains snake position (in an array), length of the snake and the direction toward snake faces. The second main variable is an array which contains the game board. Third main variable is a coordinate struct storing the currently placed apple’s position on the board.
Game environment setup starts by initializing variables. Then the terminal window size is read. Number of rows is necessary because the cursor can not move around on the screen. The count of new line prints are adjusted so the board is displayed at the same position. In the last step the terminal environment adjusted. Default terminal operation inappropriate for interactive animations. Showing of entered keys (echoing) is turned off, waiting time for keyboard entry set to zero (for continuous animation) and terminal set to raw mode so the arrow keys can be read.
After initial setup, the main program loop started. The loop consists of two parts. In first part the keyboard entry read. If the escape key pressed the exit condition set. Then according to arrow key press the snake facing direction changed. The second part is a timed loop. It contains the snake movement calculations and drawing on screen. Exact timing defines the difficulty of the game. More time the loop runs in a second, the faster the snake moves.
Timed loop begins with moving the snake forward. Snake position is stored in an array. Its size is same as game board. Each array element stores one position on the board which is occupied by the snake. Head and tail variables point to two elements in the array. When the snake moves forward the head variable increased to point at the next element in the array and new coordinate stored there. Tail also moves forward in similar manner. The array is handled as wrap around, when head or tail reach the last element they continue from the beginning.
After the head moved forward, the position is checked under the head. If it is occupied by a snake segment or wall, it considered collision and the exit condition set meaning game over. If an apple found there, the apple removed and score increased.
Following collision check, if the apple has been eaten, a new one must be placed. Apple should placed on empty position only, therefore candidate position is compared to snake segments’ position.
Then the snake and apple placed on the beforehand emptied board. The snake can not be drawn directly on screen, because the terminal window drawn in left-to-right, top-to-bottom direction. Putting game objects’ position in a random access buffer makes rendering easier.
The timed loop ends by drawing the game screen, score, board and game information on screen.
When the game finishes the terminal restored to its initial state.
The program ends with a top list. The last position on the list compared with the earned score to decide player’s position. If the player earned a place in the Top 10, a name is asked and placed on the list.

The top list is read and written back to a binary file. If the file have not existed before it is created. Finally the top list presented and the program exits.

Additional libraries

In addition to the stdio.h library the following standard and Linux libraries have used:

stdlib.h	for random number generation
string.h	for memory operations (copying, erasing)
sys/ioctl.h, unistd.h	for ioctl and read
termios.h	for reading and writing terminal parameters
time.h	for timing functions
snake.h	own library, it contains the snake handling definitions and functions

Development issues and personal achievements

The only thing I found difficult is debugging. It was sometimes challenging especially when errors raised occasionally only. Functions receiving and working on pointers also made data change following difficult.
I choose snake game because it seemed relatively easy. I intended to give plenty of time to elaborate fine details of the game and enhance user experience. I insisted to make the game interactive with easy snake control. The aim was to make less, but possibly error free and stable code with extra features.
During the development I have acquired basic insight to Linux programming. I have also read and practiced advanced C programming, like pointers, self-referencing structs, multiple file source code, binary file handling and Preprocessor directives.
I think the raw mode terminal handling is cool. It is not much, but still greatly enhance user experience.

Conclusion

The project may be regarded successful. Many aspects of project development has been practiced and almost everything about C language has been studied.

Bibliography

1. Various editors. Termios(3) - Linux man page.
   Available from: https://linux.die.net/man/3/termios [cited October 12, 2018].

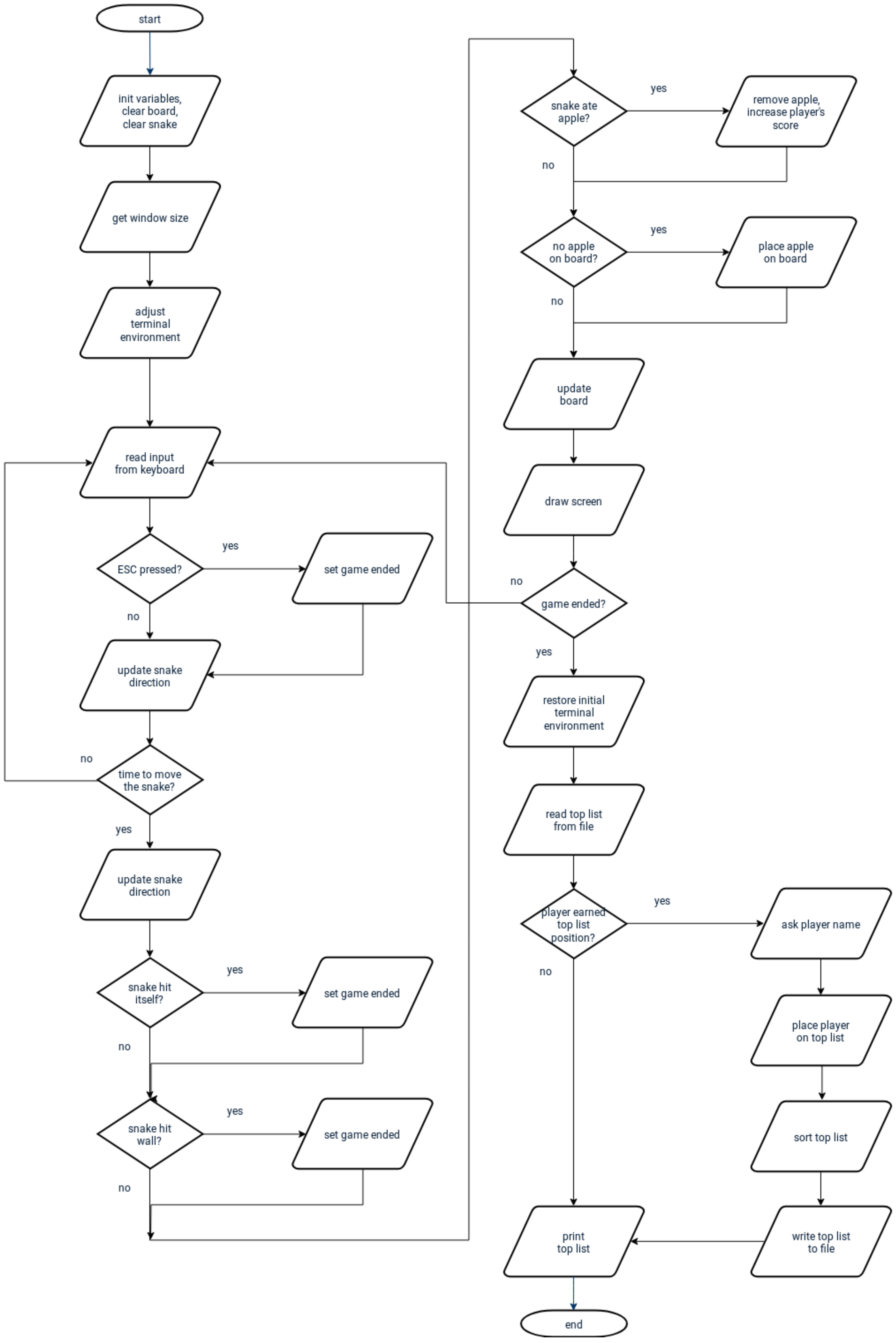
2. Various editors. Ioctl(2) - Linux man page.
   Available from: https://linux.die.net/man/2/ioctl [cited October 12, 2018].

3. Various editors. Tty_ioctl(4) - Linux man page.
   Available from: https://linux.die.net/man/4/tty_ioctl [cited October 12, 2018].

4. Various editors. Snake (video game genre).
   Available from: https://en.wikipedia.org/wiki/Snake_(video_game_genre) [cited October 12, 2018].

Appendices

Flowchart



Self-evaluation

I think I have written a well structured, well commented, functional code which is easily understandable and extendable.
I would like to improve my Linux programming skills.
Regarding the C programming language I have not covered the function pointers yet. Also Double / Triple Pointers should be further experimented with.