# Report for CSC3150 assignment 2

***Name: 舒欣***

***Student ID: 118020362***

## ***How did I design my program? Where did I place the mutex lock and why?***

First, I use global variable frog_x, frog_y to represent frog's position instead of Node struct that the template give us, since I find the use of pointer to struct to pass parameter will cause some trouble that I can't handle. Next, I set LOSE_FLAG, WIN_FLAG and isQuit global variables to signify the status of the game, and only when the game is not lose and not win and not quit can the game continue. In logs_move() function, log_num is the variable for the x-position of a log, for example, the two banks' x-position are 0 and ROW, respectively. Totally there are nine logs apart from two banks. logs_move() function is to control the specified line of log and frog's move. Inside I use srand(clock()) to generate random seed, and use rand()%(COLUMN-1) to generate the starting position of the log. Then we start the infinite loop for log and frog's move. First, we check the keyboard hit and change the frog's position according to ketboard hit, at the meantime check whether frog is out of the scope of the map[ROW+10][COLUMN].

When checking the keyboard, we need to set mutex, since we want to change and use frog's position correctly. Otherwise, other threads could use the frog's position before current thread rewrite frog's position, causing updating error. Next, for nine logs, we first draw the log using 15'='. If frog is on this log, after checking frog has not jumped into the river of this line, we can draw frog's y position as ‘0’. To check that frog has not jumped into the river, we use three conditions (tiaojian1, tiaojian2, tiaojian3), which stands for the lose case of log totally displayed between the screen ( 0    ================    0 ), and across the screen (=====     0    =========). After draw log and frog, we are able to print out this line. After printing out, we remove log and frog, and update the starting point of log, and frog’s position. If the log is left move, we decrease the starting point of log and forg’s y-position. Similar for right-move log. Again, during the drawing, printing and updating of log and frog, we need to set mutex since during which it will change frog’s position and cursor’s position, which we want to change and use correctly and display intactly, as said before. Lastly, for the two banks’ line (0 and ROW rows), we need to display the move and disappear of ‘0’ on *ROW* row (starting bank) and the appear of ‘0’ on *0* row (ending bank) (also need to remove the ‘0’ on *1* row ) when win. During this we also need to set mutex because we don’t want the cursor to move to other threads in the middle way.

Finally, in main function, I create 10 threads, nine for logs and one for the starting bank. (when win, one of threads will change the ending bank). After that, I initiate the mutex, and let the main() to wait for 10 side threads to finish, finally print out the game status message.

***<u>What problems I met in this assignment and what is the solution?</u>***

There are many problems that I encounter in this assignment.

P1. Pointer cannot work properly, I choose to use global variable. Originally, I construct a new struct containing line number of log and frog’s position to pass the parameter to threads, however after using it, there are randomly some logs cannot be properly displayed, after that I change to use global variable, it works fine. I still don't know why.

P2. Whether I should first check keyboard hit or first update frog's position.

this problem actually doesn't matter much. At least I have tried two of them, there is no big difference.

P3. Whether I should first update the frog's position or print out the log and frog.

If I first update position then print out, it is easier for player since they do not need to predict frog's position and target log's position, since what you see is what it really is. However, if I update after print out, what you see is not what it really is, it requires the player to predict both frog's position and log's position to move correctly. Knowing the difference, I choose the second one.

P4. It takes me a while to think out the condition to check whether a frog has jumped into water.

Just as I have said before, since log can intactly lay between the screen, but also can lay across the screen. Therefore, use the condition (frog_y>=(start+15)%(COLUMN-1)) || (frog_y<start%(COLUMN-1)) to check lose is not enough, it should not in the case of (start%(COLUMN-1)>(start+15)%(COLUMN-1)). Put it simple, it is like 0    =============, and ============    0 are lose, but not ===0===         =========. After that, I run the program and find out there is another lose case, which is ======    0

=========, so I add the condition frog_y>=(start+15)%(COLUMN-1)&&frog_y<start%(COLUMN-1).

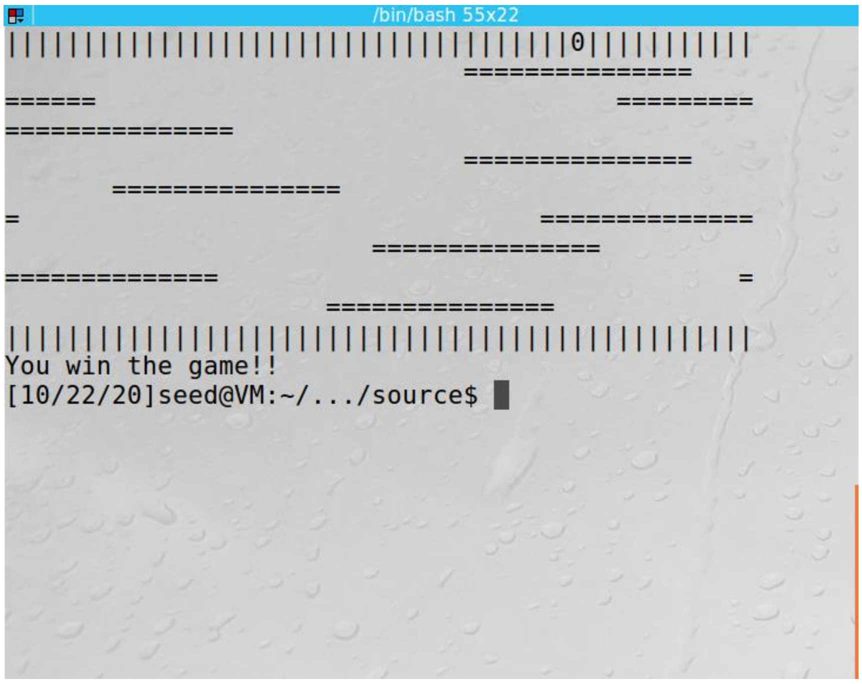
***Environment of running my program and steps to run my program***

OS: Ubuntu 16.04 (32-bit)

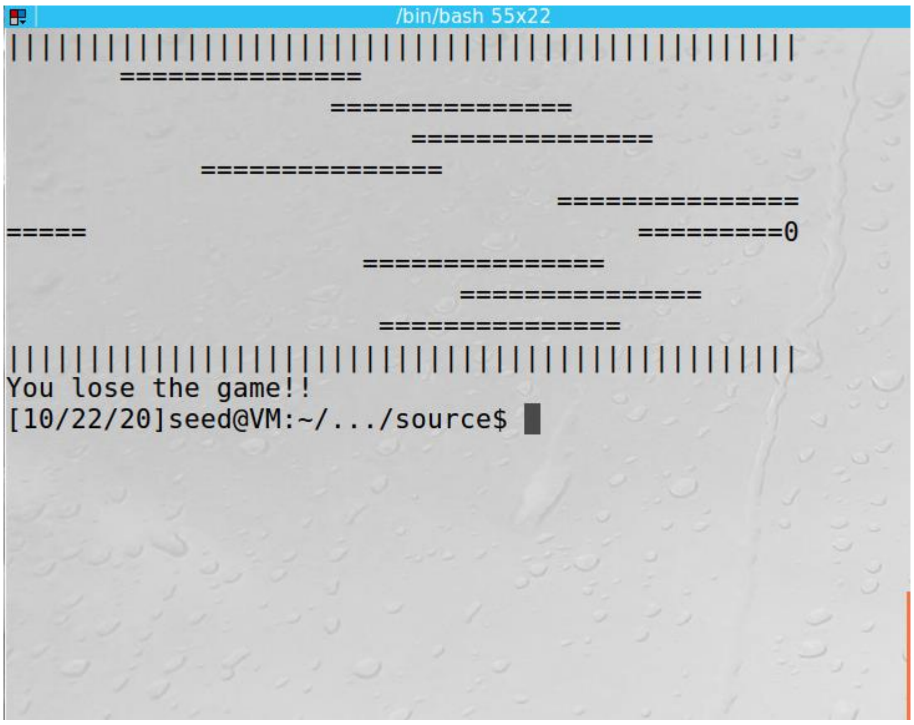
Kernel: linux-4.10.14

Steps: if you are using virtual machine, first, before open the virtual machine, you should go to Oracle VM VirtualBox Manager-> SEEDUBUNTU-> Screenshots-> Settings-> System-> number of Cores, and set the number of cores to 6. Then you go to the directory of my program, type in gcc hw2.cpp -lpthread, if it compiles successfully, type in ./a.out, you will see the output.

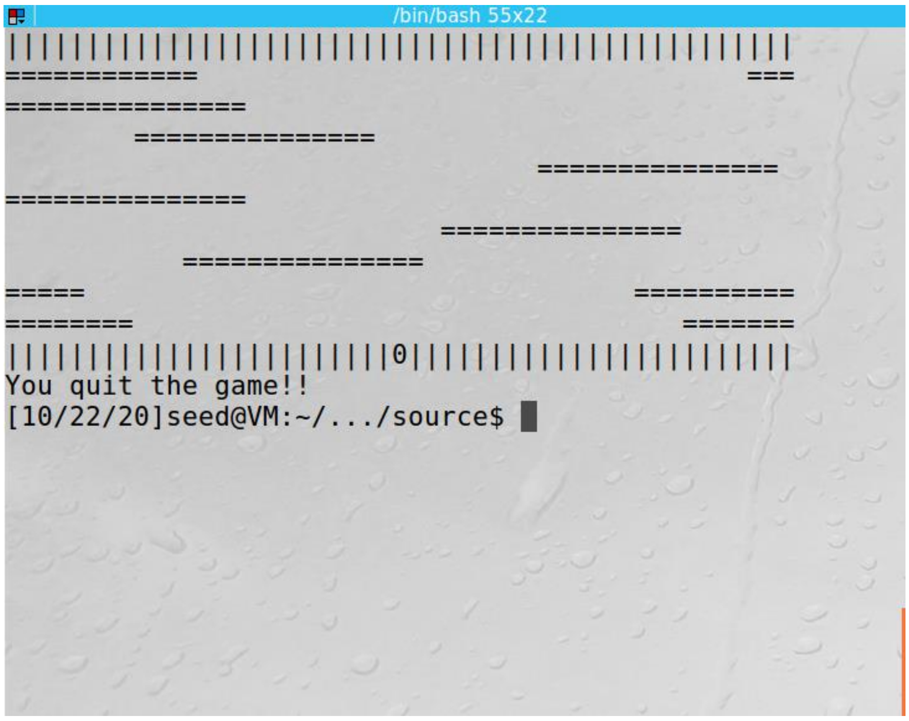
***Screenshots of my program***



```
|||||||||||||||||||||||||||||||||||||0|||||||||||
                              ===============
======                                  =========
===============
                              ===============
       ===============
=                                  ==============
                        ===============
==============                                  =
                     ===============
|||||||||||||||||||||||||||||||||||||||||||||||||
You win the game!!
[10/22/20]seed@VM:~/.../source$ 
```



```
|||||||||||||||||||||||||||||||||||||||||||||||||
       ===============
                    ===============
                         ===============
            ===============
                                  ===============
=====                                  =========0
                      ===============
                            ===============
                       ===============
|||||||||||||||||||||||||||||||||||||||||||||||||
You lose the game!!
[10/22/20]seed@VM:~/.../source$ 
```



```
|||||||||||||||||||||||||||||||||||||||||||||||||
============                                  ===
===============
        ===============
                                 ===============
===============
                           ===============
           ===============
=====                                  ==========
========                                  =======
||||||||||||||||||||||||0||||||||||||||||||||||||
You quit the game!!
[10/22/20]seed@VM:~/.../source$ 
```

***<u>What I learned in this assignment?</u>***

1. Sometimes the existence of bugs in a game just results from the difference of the understanding of the game rules between programmers and players.

2. Multi-thread programming is very important and useful.
3. There are lots of details you need to consider when designing and writing a game.