Assignment Report

CSC 3150 Multi-Thread Programming

Wei Wu (吴畏)

118010335

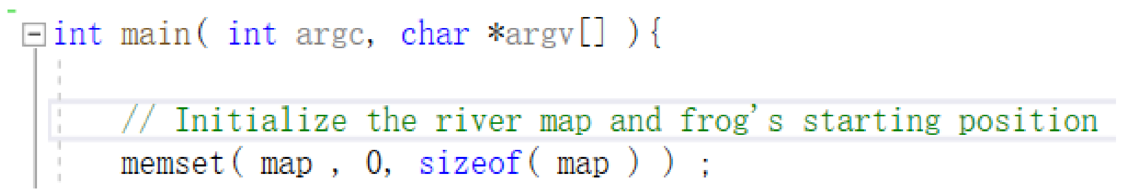
October 24, 2020

The School of Data Science

香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

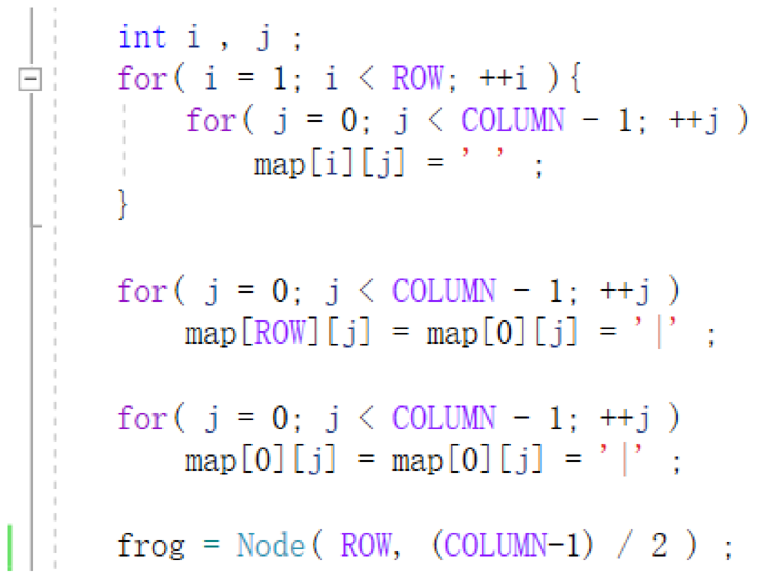
1. How Did I Design My Program

a. In main(), allocate memory for the map structure.



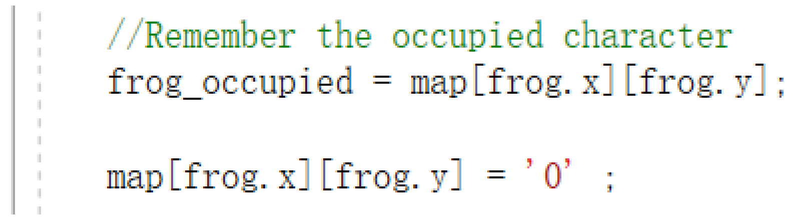
```
int main( int argc, char *argv[] ){

    // Initialize the river map and frog's starting position
    memset( map , 0, sizeof( map ) ) ;
```

b. Initialize the river and the frog.



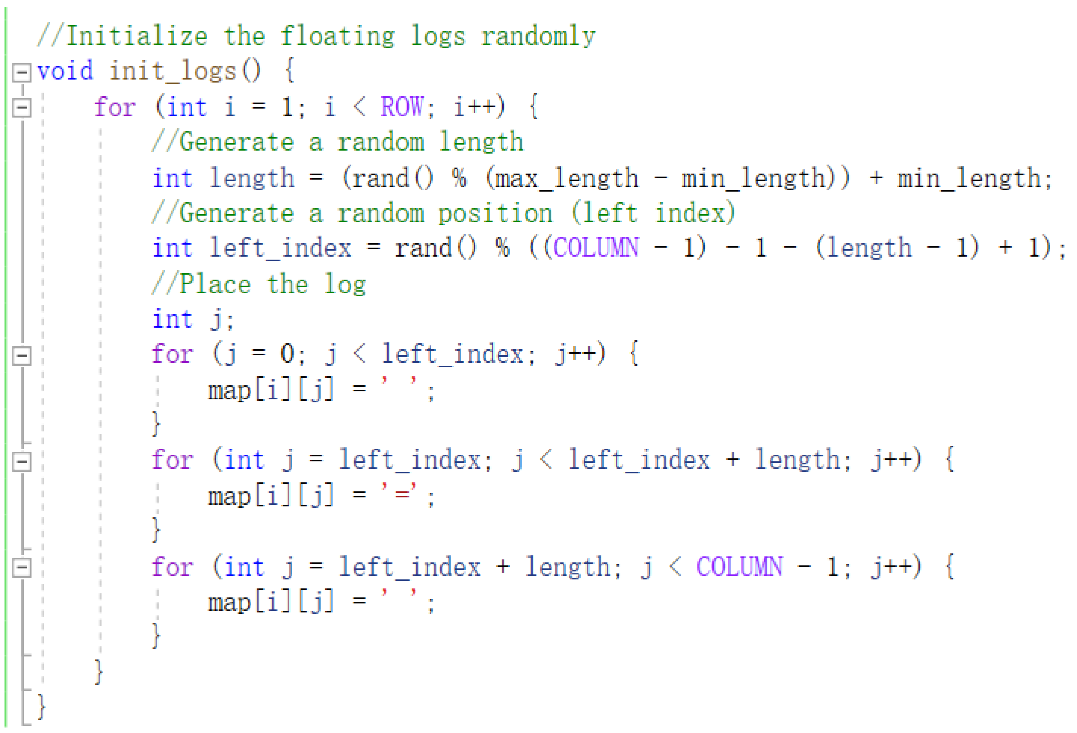
```
    int i , j ;
    for( i = 1; i < ROW; ++i ){
        for( j = 0; j < COLUMN - 1; ++j )
            map[i][j] = ' ' ;
    }

    for( j = 0; j < COLUMN - 1; ++j )
        map[ROW][j] = map[0][j] = '|' ;

    for( j = 0; j < COLUMN - 1; ++j )
        map[0][j] = map[0][j] = '|' ;

    frog = Node( ROW, (COLUMN-1) / 2 ) ;
```

c. Initialize the "frog_occupied" variable, which remembers the character occupied by the frog. Then place the frog.



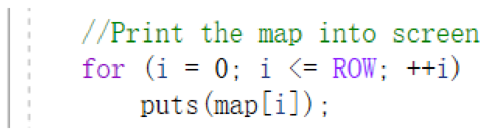
```
    //Remember the occupied character
    frog_occupied = map[frog.x][frog.y];

    map[frog.x][frog.y] = '0' ;
```

d. Call init_logs() to initialize the logs. The lengths and positions of the logs will be generated randomly.



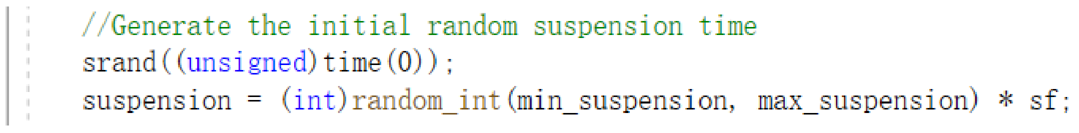
```
//Initialize the floating logs randomly
void init_logs() {
    for (int i = 1; i < ROW; i++) {
        //Generate a random length
        int length = (rand() % (max_length - min_length)) + min_length;
        //Generate a random position (left index)
        int left_index = rand() % ((COLUMN - 1) - 1 - (length - 1) + 1);
        //Place the log
        int j;
        for (j = 0; j < left_index; j++) {
            map[i][j] = ' ';
        }
        for (int j = left_index; j < left_index + length; j++) {
            map[i][j] = '=';
        }
        for (int j = left_index + length; j < COLUMN - 1; j++) {
            map[i][j] = ' ';
        }
    }
}
```

e. Print the map into the screen.



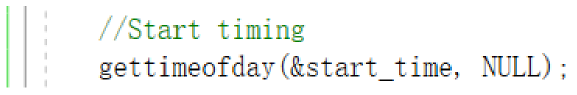
```
//Print the map into screen
for (i = 0; i <= ROW; ++i)
    puts(map[i]);
```

f. Generate the initial random suspension time.



```
//Generate the initial random suspension time
srand((unsigned)time(0));
suspension = (int)random_int(min_suspension, max_suspension) * sf;
```

g. Start timing.



```
//Start timing
gettimeofday(&start_time, NULL);
```

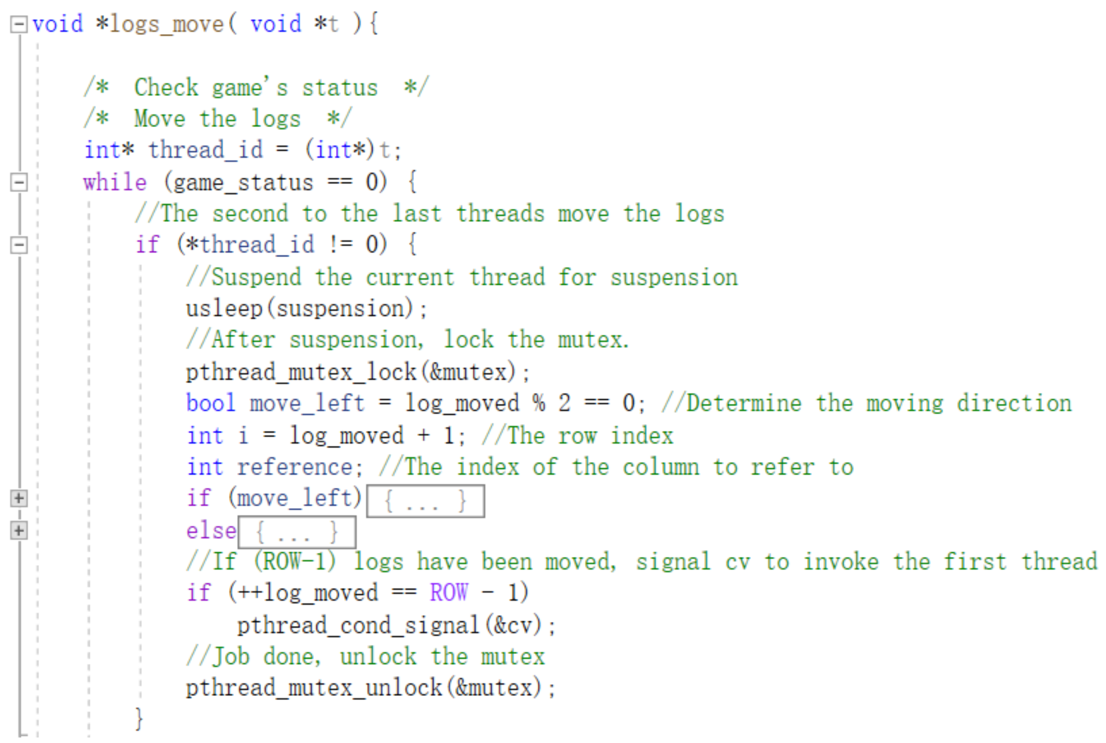
h. Declare and initialize the pthreads, pthread attribute, mutex, condition variable.

```
/*  Create pthreads for wood move and frog control.  */

//Declare the threads
pthread_t threads[10];

//Initialize the attribute
pthread_attr_t attr;
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

//Initialize the mutex and condition variable
pthread_mutex_init(&mutex, NULL);
pthread_cond_init(&cv, NULL);

//Create threads
for (int i = 0; i < THREAD; i++)
    pthread_create(&threads[i], &attr, logs_move, (void*)&thread_ids[i]);
```

i. Join the threads for synchronization.

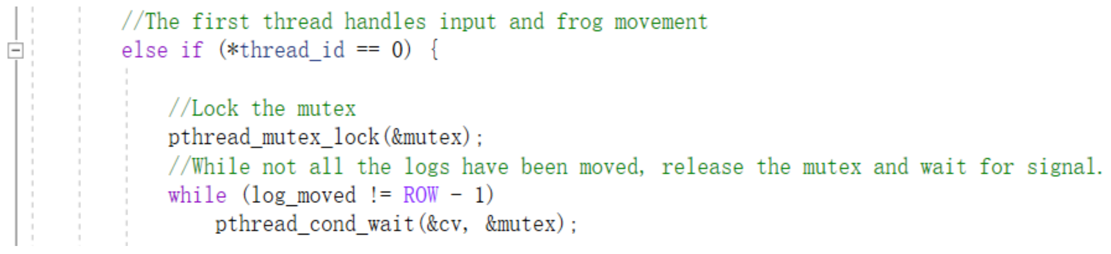
```
//Join the threads
for (int i = 0; i < THREAD; i++)
    pthread_join(threads[i], NULL);
```

j. In the function logs_move, while the game is not over, the second to the last threads move the logs in turn. (One thread moves one log at a time.) When a thread is invoked, it first suspends for a randomly generated time. Then, after the suspension, it will move a log. Since moving a log requires access to shared data (the map), the thread must lock the mutex before the movement and unlock the mutex after the movement. In addition, if the count of logs that have been moved in this round equals the total number of logs (ROW – 1), the thread should send a condition signal to invoke the first thread.



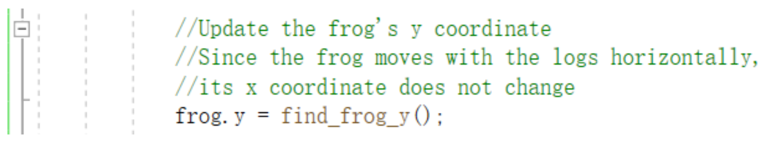
```
void *logs_move( void *t ){

    /*  Check game's status  */
    /*  Move the logs  */
    int* thread_id = (int*)t;
    while (game_status == 0) {
        //The second to the last threads move the logs
        if (*thread_id != 0) {
            //Suspend the current thread for suspension
            usleep(suspension);
            //After suspension, lock the mutex.
            pthread_mutex_lock(&mutex);
            bool move_left = log_moved % 2 == 0; //Determine the moving direction
            int i = log_moved + 1; //The row index
            int reference; //The index of the column to refer to
            if (move_left) { ... }
            else { ... }
            //If (ROW-1) logs have been moved, signal cv to invoke the first thread
            if (++log_moved == ROW - 1)
                pthread_cond_signal(&cv);
            //Job done, unlock the mutex
            pthread_mutex_unlock(&mutex);
        }
```

k. The first thread firstly locks the mutex. While not all the logs have been moved, it releases the mutex and waits for signal through pthread_cond_wait(). This step enables the other threads to access and modify the shared data, that is, to move the logs.



```
        //The first thread handles input and frog movement
        else if (*thread_id == 0) {

            //Lock the mutex
            pthread_mutex_lock(&mutex);
            //While not all the logs have been moved, release the mutex and wait for signal.
            while (log_moved != ROW - 1)
                pthread_cond_wait(&cv, &mutex);
```

l. When the condition signal is received, it updates the frog's y coordinate. Since the frog moves with the logs horizontally, its x coordinate does not change.



```
            //Update the frog's y coordinate
            //Since the frog moves with the logs horizontally,
            //its x coordinate does not change
            frog.y = find_frog_y();
```

m. Then it generates a new random suspension time and reset the log_moved variable for the next round.

```
            //Generate a new random suspension time
            suspension = (int) random_int(min_suspension, max_suspension) * sf;

            //Reset log_moved
            log_moved = 0;
```

n. Next, it checks keyboard hits. If there is, it calls input_event() to handle the input.

```
            /* Check keyboard hits, to change frog's position or quit the game. */
            //Handle the input
            if (kbhit()) {
                char key = getchar();
                input_event(key);
            }
```

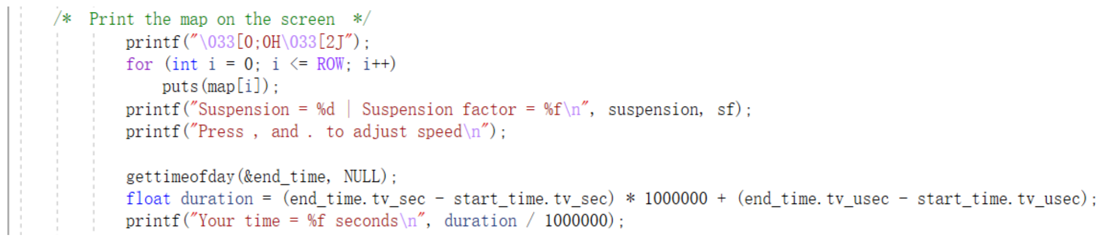
```
void input_event(char key) {
    //Quit
    if (key == 'q' || key == 'Q'){ ... }
    //Adjust speed
    //Speed up
    else if (key == '.'){ ... }
    //Speed down
    else if (key == ','){ ... }
    //Move
    else if (key == 'w' || key == 'W'){ ... }
    else if (key == 's' || key == 'S'){ ... }
    else if (key == 'a' || key == 'A'){ ... }
    else if (key == 'd' || key == 'D'){ ... }
}
```

o. If the game status till now is normal, it updates the game status by calling check_status().

```
            //Update the game status
            if (game_status == 0)
                game_status = check_status();
```

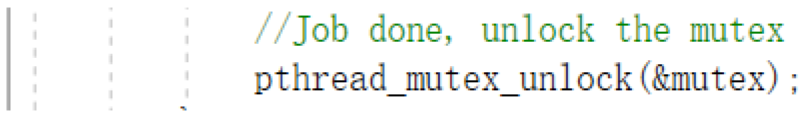
```
int check_status() {
    if (frog.x == 0)
        return 1;
    else if (frog_occupied == '=' || frog_occupied == '|')
        return 0;
    else if (frog_occupied == ' ')
        return 2;
}
```

p. Print the map onto the screen.



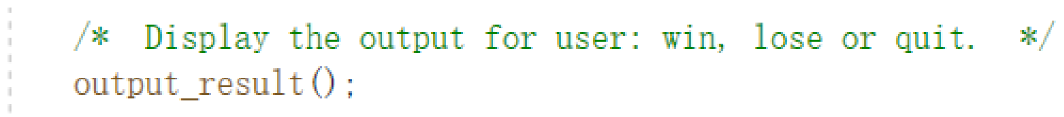
```
/*  Print the map on the screen  */
    printf("\033[0;0H\033[2J");
    for (int i = 0; i <= ROW; i++)
        puts(map[i]);
    printf("Suspension = %d | Suspension factor = %f\n", suspension, sf);
    printf("Press , and . to adjust speed\n");

    gettimeofday(&end_time, NULL);
    float duration = (end_time.tv_sec - start_time.tv_sec) * 1000000 + (end_time.tv_usec - start_time.tv_usec);
    printf("Your time = %f seconds\n", duration / 1000000);
```

q. Unlock the mutex since the job is done.

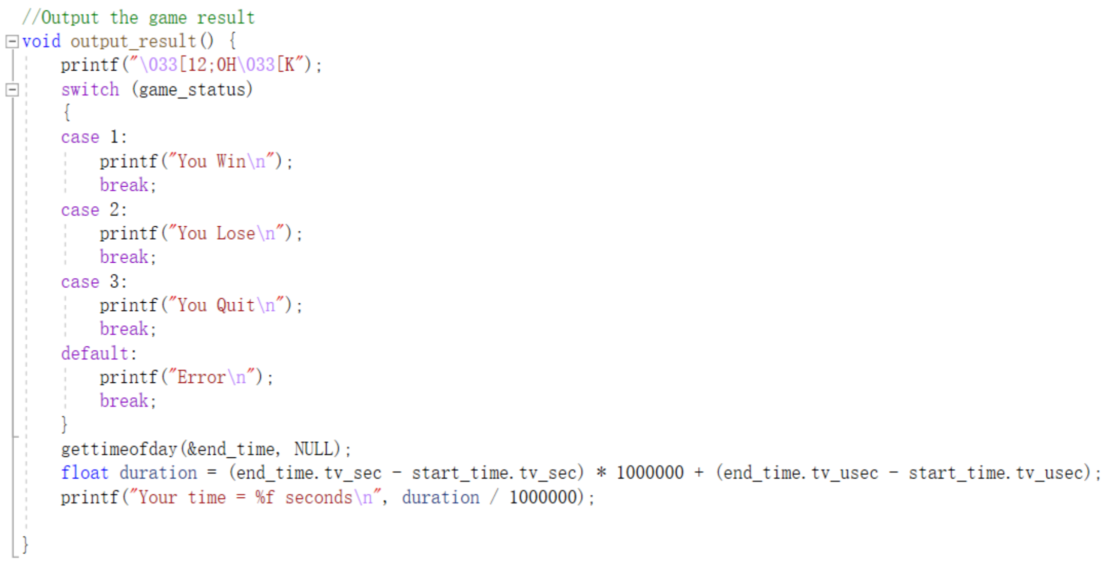


```
//Job done, unlock the mutex
pthread_mutex_unlock(&mutex);
```

r. If the game status indicates that the game is over, the threads will exit. And then main() will call output_result() to output the game result.

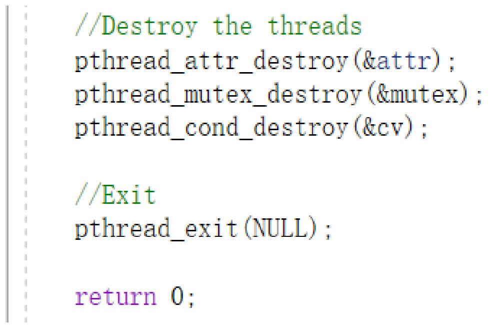


```
/*  Display the output for user: win, lose or quit.  */
output_result();
```



```
//Output the game result
void output_result() {
    printf("\033[12;0H\033[K");
    switch (game_status)
    {
    case 1:
        printf("You Win\n");
        break;
    case 2:
        printf("You Lose\n");
        break;
    case 3:
        printf("You Quit\n");
        break;
    default:
        printf("Error\n");
        break;
    }
    gettimeofday(&end_time, NULL);
    float duration = (end_time.tv_sec - start_time.tv_sec) * 1000000 + (end_time.tv_usec - start_time.tv_usec);
    printf("Your time = %f seconds\n", duration / 1000000);

}
```

s. Main() destroys the pthread stuff and exit.

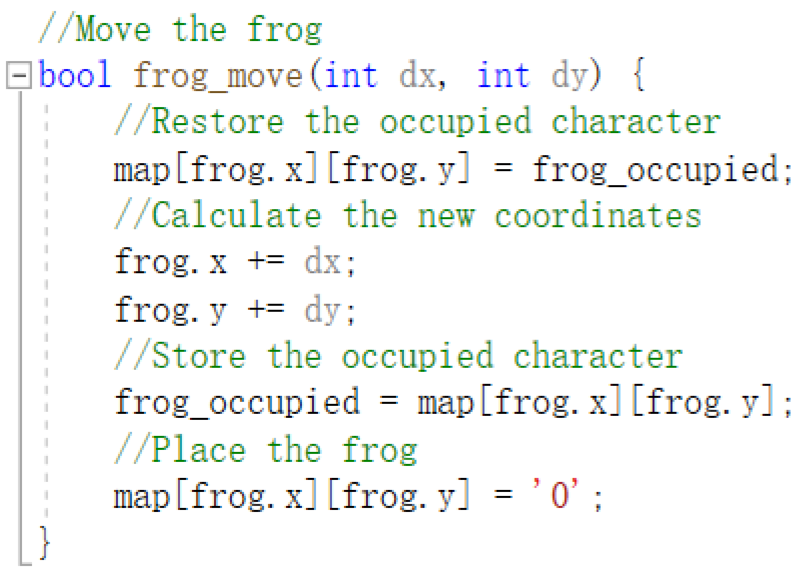


```
//Destroy the threads
pthread_attr_destroy(&attr);
pthread_mutex_destroy(&mutex);
pthread_cond_destroy(&cv);

//Exit
pthread_exit(NULL);

return 0;
```

2. What Problems I Met in This Assignment and What is My Solution?

a. What character is currently occupied by the frog? i.e. What is the frog currently standing on?

I declared a global variable `char frog_occupied` to remember the current character occupied by the frog. If the frog is on the logs, when the logs move, the frog moves with them. Hence, the occupied character will not change. While if the player moves the frog through WASD, the occupied character will change. So, I update the character occupied on each valid frog movement by player.



```
//Move the frog
bool frog_move(int dx, int dy) {
    //Restore the occupied character
    map[frog.x][frog.y] = frog_occupied;
    //Calculate the new coordinates
    frog.x += dx;
    frog.y += dy;
    //Store the occupied character
    frog_occupied = map[frog.x][frog.y];
    //Place the frog
    map[frog.x][frog.y] = '0';
}
```

b. How to generate a random suspension time in each round?

I implemented a simple function to generate a random integer in range (min, max) as a tool. If the random suspension time is generated in each round in each of the threads, it will be different for each thread, which is messy. Therefore, I generate the random suspension time in each round but only in the first thread, so that suspension will be the same for every thread.

c. How to adjust the game speed in run time?

Adjusting the game speed is essentially equivalent to adjusting the suspension time. Therefore, I declared a “suspension factor”, which is multiplied to the random generated suspension time, to have an overall control on the game speed.

```
//Generate a new random suspension time
suspension = (int) random_int(min_suspension, max_suspension) * sf;
```

d. How to randomize the length of each log?

I generated the log lengths within a range (min_length, max_length).

```
//Log parameters
int min_length = 8;
int max_length = 15;
```

e. How to trace the game status?

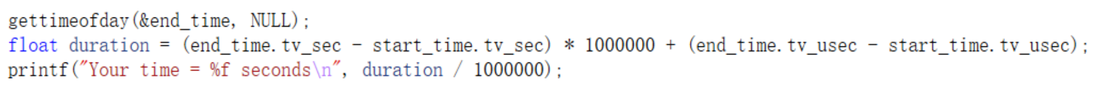
I used a integer “game_status” to trace the current status.

```
//Indicate the current game status
//0: normal
//1: win
//2: lose
//3: quit
int game_status = 0;
```

f. How to keep track of the game time?

I used the structures and functions provided by <sys/time.h>.

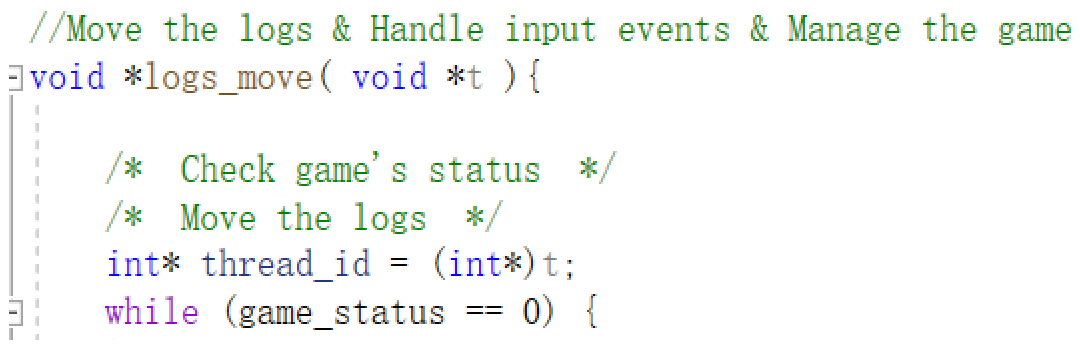
```
//Start timing
gettimeofday(&start_time, NULL);
```



```
gettimeofday(&end_time, NULL);
float duration = (end_time.tv_sec - start_time.tv_sec) * 1000000 + (end_time.tv_usec - start_time.tv_usec);
printf("Your time = %f seconds\n", duration / 1000000);
```

g. How to exit the threads (stop the game) when the game is over?

With the help of the global variable "game_status", I wrote a while loop for each thread. When game_status is not 0, the threads shall exit.



```
//Move the logs & Handle input events & Manage the game
void *logs_move( void *t ){

    /*  Check game's status  */
    /*  Move the logs  */
    int* thread_id = (int*)t;
    while (game_status == 0) {
```

3. The Environment of Running My Program

OS: Linux version 4.10.14 (root@VM)

g++ version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.4)

4. The Steps to Execute My Program

CONTENTS:

hw2.cpp

README

HOW TO COMPILE:

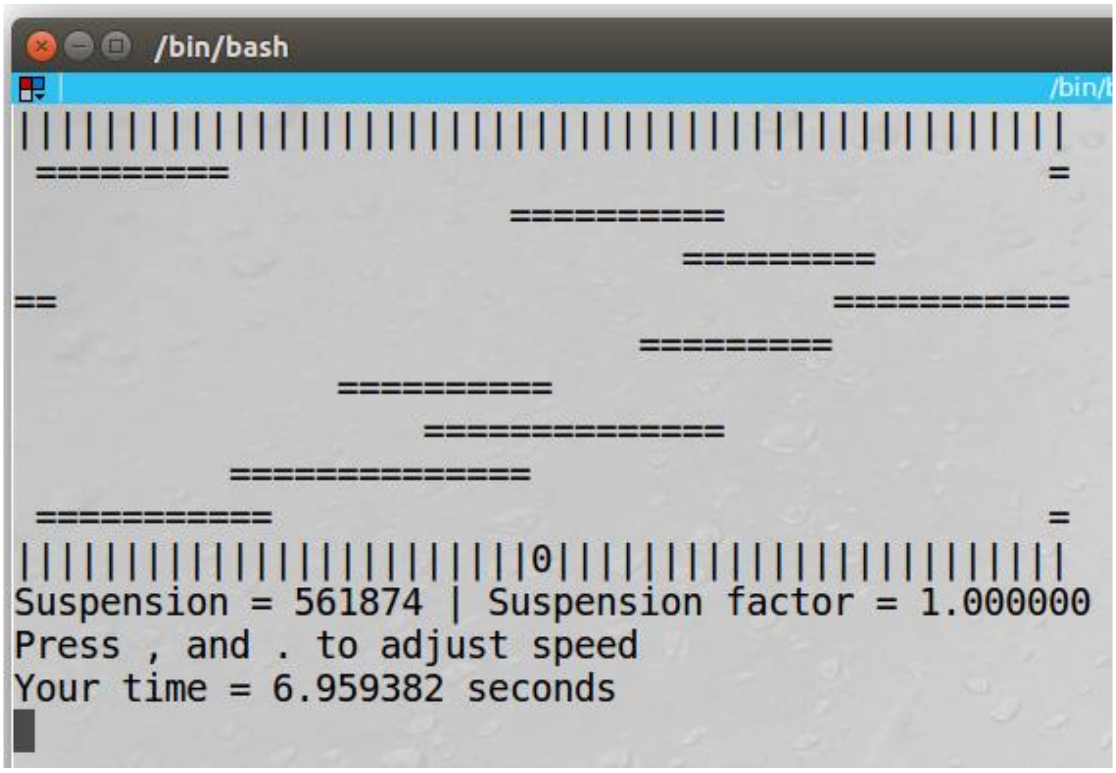
In the 'source' directory, type 'g++ hw2.cpp -lpthread' and enter on concole.

HOW TO EXECUTE:

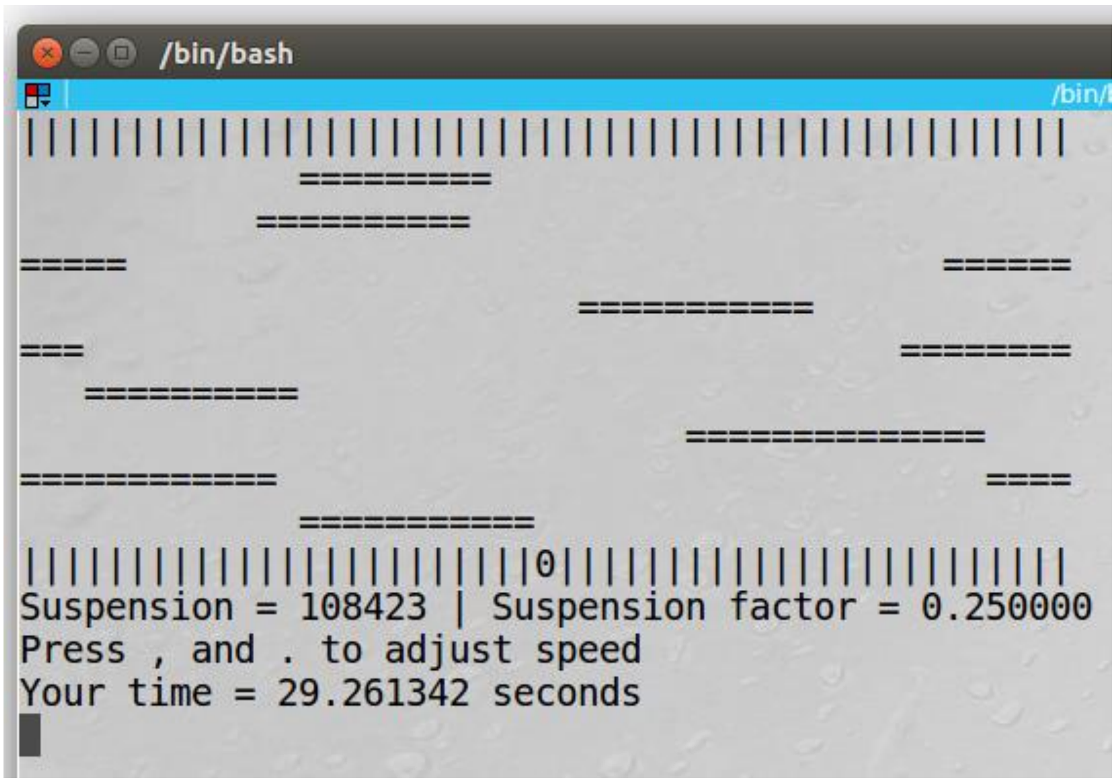
In the 'source' directory, type './a.out',

Note: To adjust the game speed, press ',' to slow down and '.' to speed up.

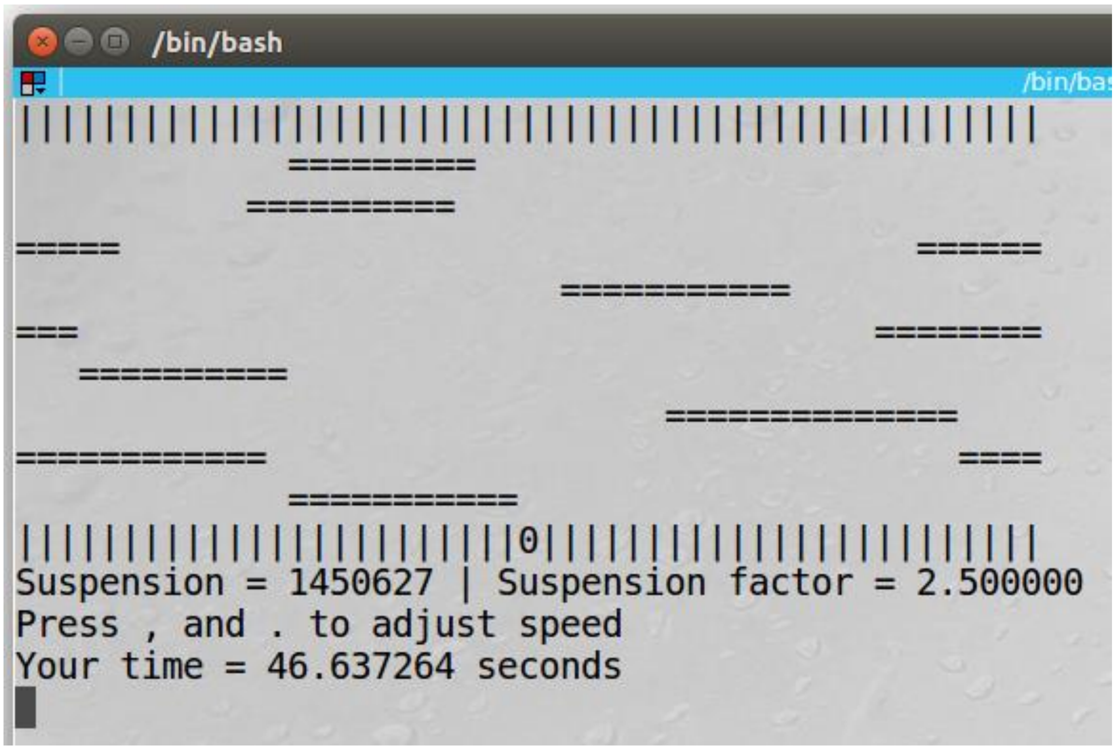
5. Screenshots of My Program Output



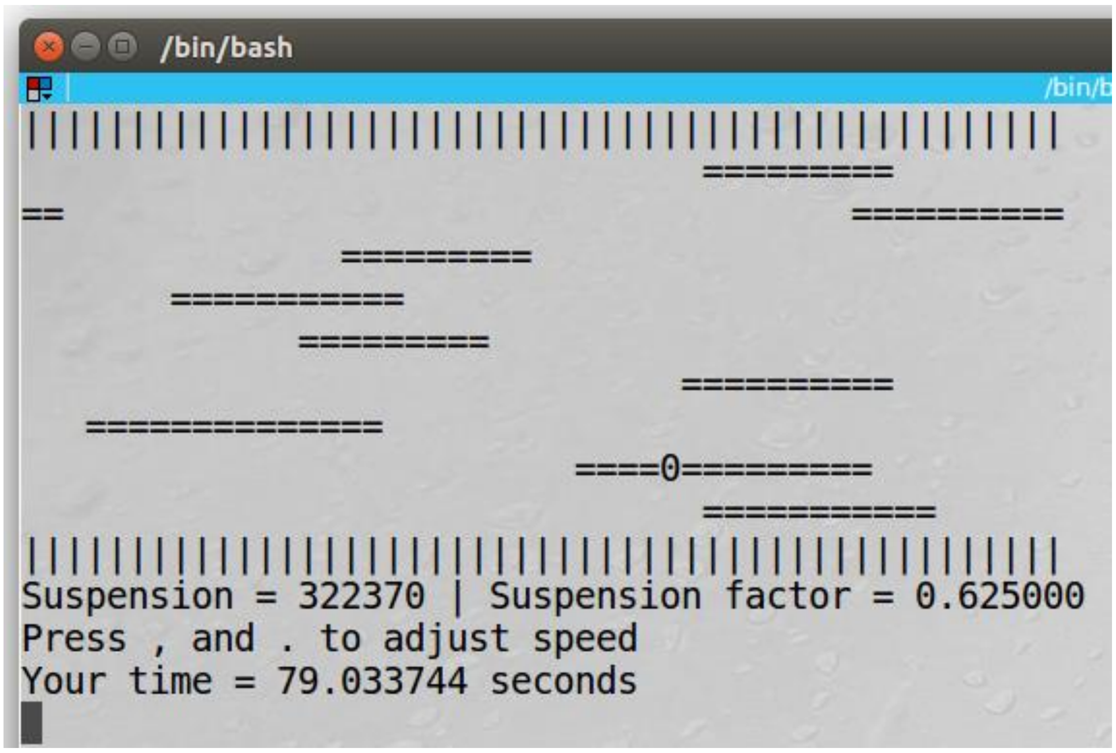
```
|||||||||||||||||||||||||||||||||||||||||||||||||
 =========                                      =
                       ==========
                               =========
==                                    ===========
                             =========
               ==========
                   ==============
          ==============
 ===========                                    =
||||||||||||||||||||||||0||||||||||||||||||||||||
Suspension = 561874 | Suspension factor = 1.000000
Press , and . to adjust speed
Your time = 6.959382 seconds
█
```



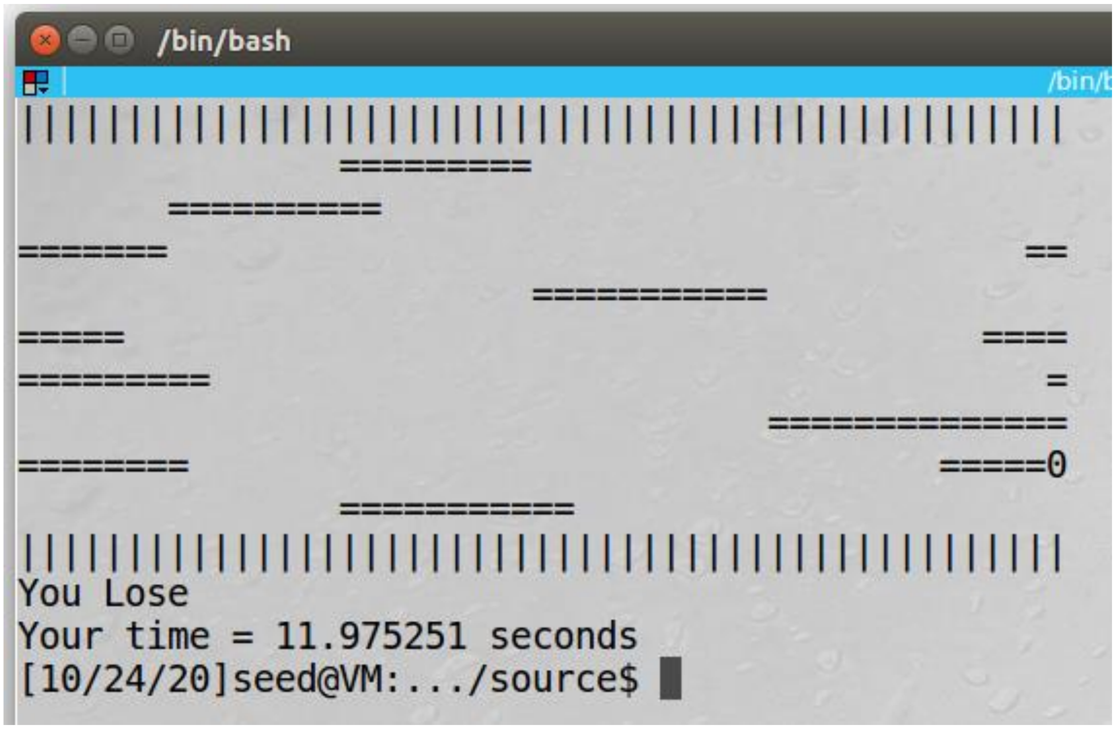
```
|||||||||||||||||||||||||||||||||||||||||||||||||
             =========
           ==========
=====                                      ======
                          ===========
===                                      ========
   ==========
                               ==============
============                                 ====
             ===========
||||||||||||||||||||||||0||||||||||||||||||||||||
Suspension = 108423 | Suspension factor = 0.250000
Press , and . to adjust speed
Your time = 29.261342 seconds
```



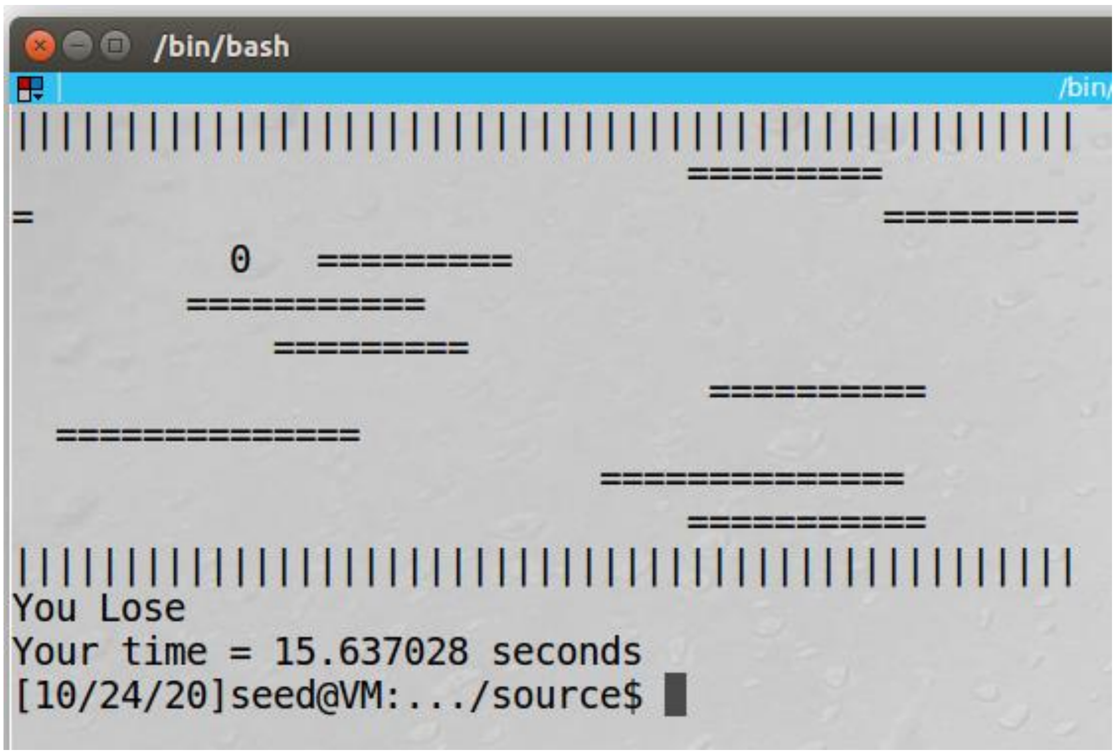
```
|||||||||||||||||||||||||||||||||||||||||||||||||
             =========
           ==========
=====                                      ======
                          ===========
===                                      ========
   ==========
                               ==============
============                                 ====
             ===========
||||||||||||||||||||||||0||||||||||||||||||||||||
Suspension = 1450627 | Suspension factor = 2.500000
Press , and . to adjust speed
Your time = 46.637264 seconds
```



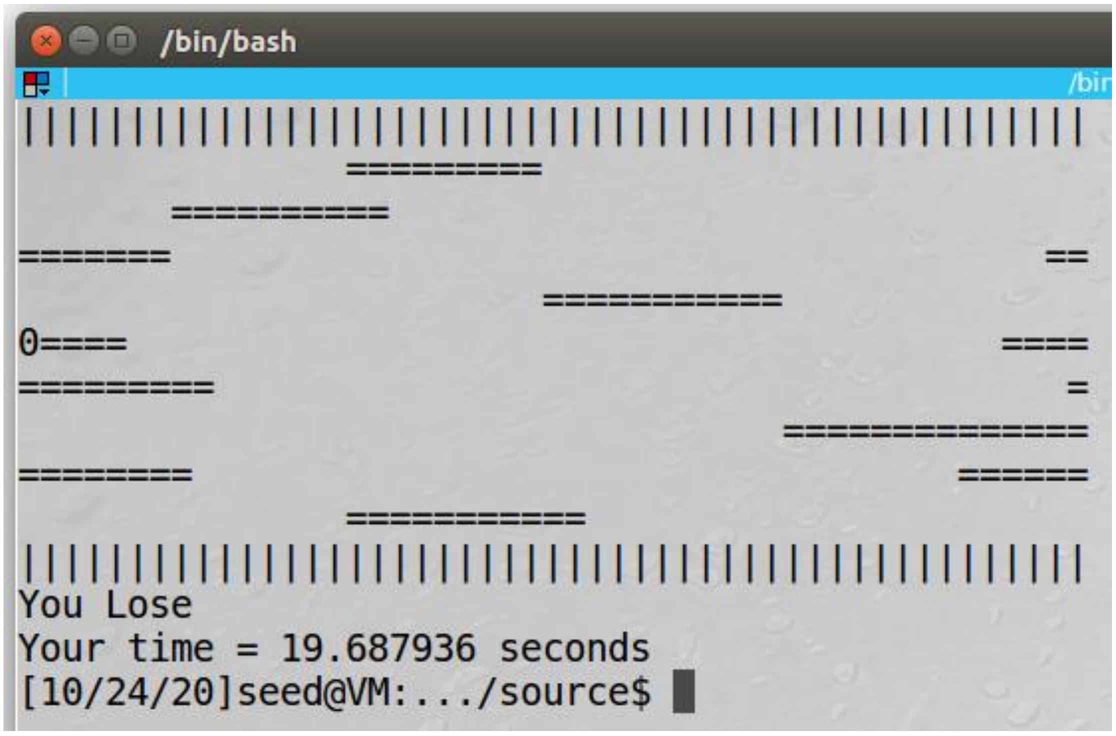
```
||||||||||||||||||||||||||||||||||||||||||||||||||
                                 =========
==                                      ==========
               =========
       ===========
             =========
                                ==========
   ==============
                           ====0=========
                                 ===========
||||||||||||||||||||||||||||||||||||||||||||||||||
Suspension = 322370 | Suspension factor = 0.625000
Press , and . to adjust speed
Your time = 79.033744 seconds
```



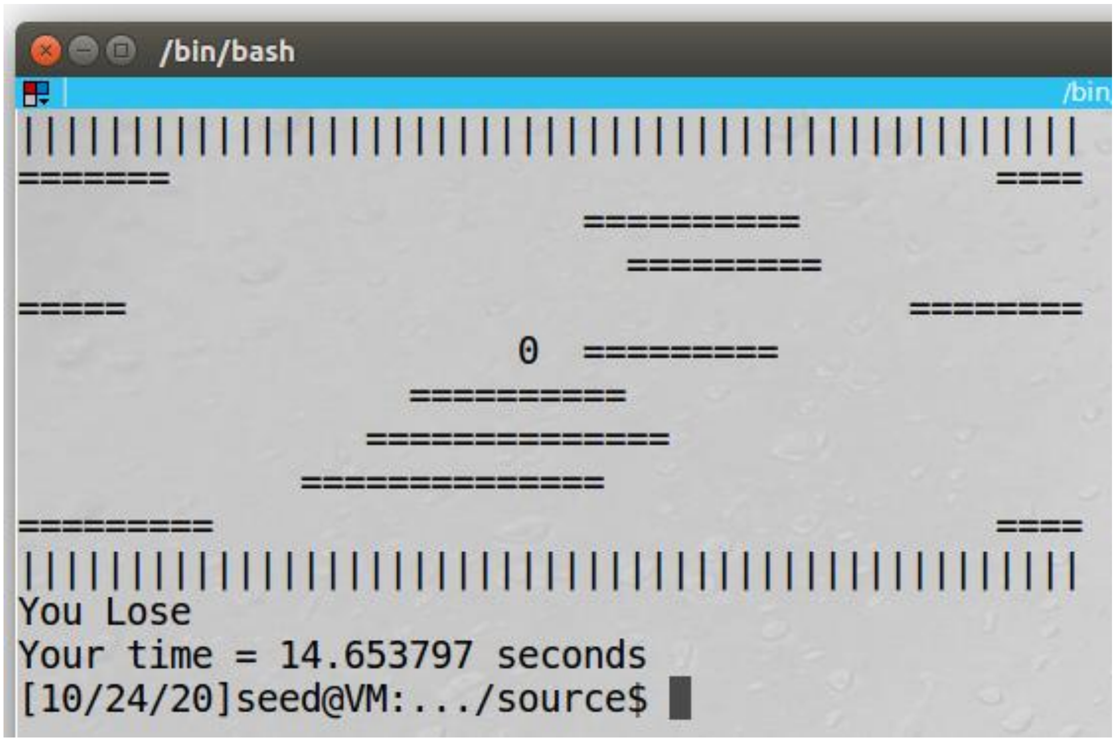
```
||||||||||||||||||||||||||||||||||||||||||||||||||
               =========
       ==========
=======                                         ==
                         ===========
=====                                         ====
=========                                        =
                                    ==============
========                                    =====0
               ===========
||||||||||||||||||||||||||||||||||||||||||||||||||
You Lose
Your time = 11.975251 seconds
[10/24/20]seed@VM:.../source$
```



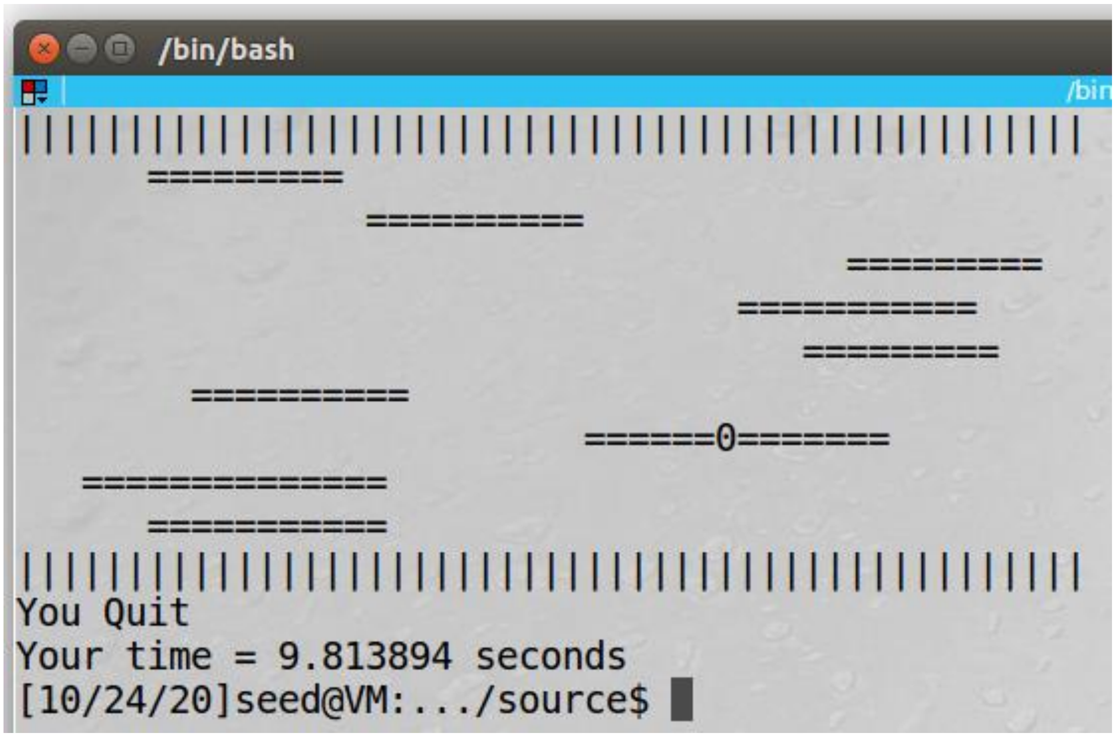
```
|||||||||||||||||||||||||||||||||||||||||||||||||
                               =========
=                                       =========
          0   =========
        ===========
            =========
                                ==========
  ==============
                           ==============
                               ===========
|||||||||||||||||||||||||||||||||||||||||||||||||
You Lose
Your time = 15.637028 seconds
[10/24/20]seed@VM:.../source$
```



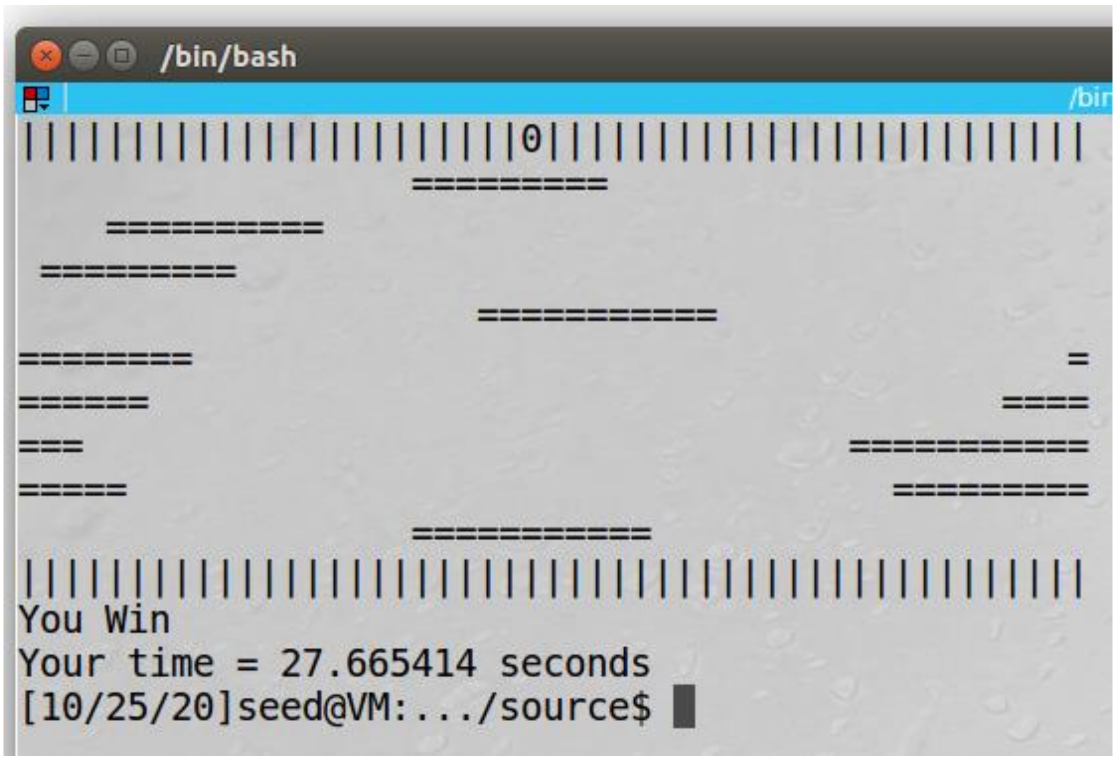
```
|||||||||||||||||||||||||||||||||||||||||||||||||
               =========
       ==========
=======                                        ==
                        ===========
0====                                        ====
=========                                       =
                                   ==============
========                                   ======
               ===========
|||||||||||||||||||||||||||||||||||||||||||||||||
You Lose
Your time = 19.687936 seconds
[10/24/20]seed@VM:.../source$
```



```
|||||||||||||||||||||||||||||||||||||||||||||||||
=======                                      ====
                          ==========
                            =========
=====                                    ========
                        0  =========
                  ==========
                ==============
             ==============
=========                                    ====
|||||||||||||||||||||||||||||||||||||||||||||||||
You Lose
Your time = 14.653797 seconds
[10/24/20]seed@VM:.../source$ 
```



```
|||||||||||||||||||||||||||||||||||||||||||||||||
      =========
                ==========
                                      =========
                                 ===========
                                    =========
        ==========
                          ======0=======
   ==============
      ===========
|||||||||||||||||||||||||||||||||||||||||||||||||
You Quit
Your time = 9.813894 seconds
[10/24/20]seed@VM:.../source$ 
```



```
|||||||||||||||||||||||0|||||||||||||||||||||||||
                  =========
    ==========
 =========
                     ===========
========                                        =
======                                       ====
===                                   ===========
=====                                   =========
                  ===========
|||||||||||||||||||||||||||||||||||||||||||||||||
You Win
Your time = 27.665414 seconds
[10/25/20]seed@VM:.../source$ 
```

6. What Did I Learn from The Tasks

a. How to initialize a pthread_attr structure.

```
//Initialize the attribute
pthread_attr_t attr;
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
```

b. How to inilialize the mutex and condition variable.

```
//Initialize the mutex and condition variable
pthread_mutex_init(&mutex, NULL);
pthread_cond_init(&cv, NULL);
```

c. How to create pthreads and let them execute the desired function.

```
//Create threads
for (int i = 0; i < THREAD; i++)
    pthread_create(&threads[i], &attr, logs_move, (void*)&thread_ids[i]);
```

d. How to join the threads for synchronization.

```
//Join the threads
for (int i = 0; i < THREAD; i++)
    pthread_join(threads[i], NULL);
```

e. How to destroy the pthread attribute, mutex, and condition variable after use.

```
//Destroy the pthread stuff
pthread_attr_destroy(&attr);
pthread_mutex_destroy(&mutex);
pthread_cond_destroy(&cv);
```

f. How to use each thread to execute the function. How to distinguish each thread using thread ID. How to suspend the thread. How and where to lock and unlock the mutex. How and when to set condition variable wait and send condition variable signal.

g. How to check keyboard hits. How to receive and handle keyboard input.

h. How to keep the time.