

CSC 3150 Project #2

Ziyu XIE
121090642

October 13, 2023

Contents

1	Introduction	1
1.1	Project tasks	1
1.2	Development environment	1
2	Implementing	1
2.1	Basic design	1
2.2	Global preparations	1
2.3	Implementing the main function	2
2.4	Implementing useful helping functions	4
2.5	Implementing the logs pthreads	5
2.6	Implementing the control threads	7
3	Executing and output	8
4	Learn from the task	12

1 Introduction

1.1 Project tasks

This project requires us to create a frog-cross-river game. A river has logs floating on it, and a frog must cross the river by jumping on the logs as they pass by via clicking "W", "S", "A", "D". This game should output "win", "loss", or "quit" status when different situations happen. To make such game in terminal using c++, We need to set the correct Linux environment and some knowledge about the multi-thread processing.

1.2 Development environment

The development environment is Ubuntu 16.04 with Linux kernel version 5.10.197. The gcc version is 5.4.0.

To set up this environment, we should first get the correct Linux kernel version installed. Follow the command lines below to get the kernel version 5.10.197.

2 Implementing

2.1 Basic design

To finish this project, we need some units like the frog and the maps. We use a 2D array to store the map, and we use a structure to store the frog. Then, we should create pthreads for logs, and we should also create some pthreads to control the screen and the game.

For the logs threads, we should let the logs to move automatically and detect whether there is a keyboard clicking. If clicking, we should process the frog moving.

For the control threads, we should refresh the screen with a high frequency and judge the game status.

The details will be shown later.

2.2 Global preparations

In this part, I firstly define the constant numbers, including the row, column, length of logs, and the delay time of log and print.

Then I define the struct for the frog, the char-2D- array-based map, the

register to record the center of each log, and the game status. In addition, there is a mutex to protect the map.

The details are shown below.

```

1  /* const numbers define */
2  #define ROW 10
3  #define COLUMN 50
4  #define LOG_LEN 15
5  #define LOG_DELAY 120000
6  #define PRINT_DELAY 13000
7
8  /* struct for the frog */
9  struct Node
10 {
11     int x, y;
12     Node(int _x, int _y) : x(_x), y(_y){};
13     Node(){};
14 } frog;
15
16 /* global variables */
17 char map[ROW + 10][COLUMN];
18 int log_center[ROW + 1];
19 int game_status; // 0 for run, 1 for win, 2 for lose, 3 for quit
20
21 /* global pthread variables */
22 pthread_mutex_t wood_lock;
23
24 /* useful helping functions */
25 int kbhit(void);
26 void map_print(void);
27 int rand_num_gen(void);
28 int get_mod(int, int);
29 /* logs helping functions */
30 void logs_shifting(int, int);
31 void logs_initial(int);
32 /* frog move helping functions */
33 void frog_move_help(char ch);
34 /* pthread functions */
35 void *screen_print(void *t);
36 void *logs_move(void *t);

```

2.3 Implementing the main function

The main functions can be divided into three parts, the initialization, the pthreads creation, and the pthreads join.

The first part is to initialize the map and status. We should use the `printf("\033[2J")`

to clean the terminal. And we should also initialize the mutex.

```

1  /* Initialize the river map and frog's starting position */
2  memset(map, 0, sizeof(map)); // set all the chars(1 byte each)
   of the map to '0'
3  for (i = 1; i < ROW; ++i) // create the empty river
4  {
5      for (j = 0; j < COLUMN - 1; ++j)
6          map[i][j] = ' ';
7  }
8  for (j = 0; j < COLUMN - 1; ++j) // create the bottom bank with
   len (COLUMN - 1)
9      map[ROW][j] = map[0][j] = '|';
10 for (j = 0; j < COLUMN - 1; ++j) // create the top bank with
   len (COLUMN - 1)
11     map[0][j] = map[ROW][j] = '|';
12 frog = Node(ROW, (COLUMN - 1) / 2); // create the frog
13 map[frog.x][frog.y] = '0'; // put the frog on the map
14
15 /* clean the terminal and set game status */
16 printf("\033[2J");
17 map_print();
18 game_status = 0;
19
20 /* Initialize the mutex for wood move */
21 pthread_mutex_init(&wood_lock, NULL);

```

The second part is to create the pthreads. We should create the pthreads for logs, the pthreads for control, and the pthreads for screen.

```

1  /* Create pthreads for wood move and frog control. */
2  pthread_t wood_threads[ROW]; // 0 for frog, others for wood
3  for (i = 1; i <= ROW - 1; i++)
4      pthread_create(&wood_threads[i], NULL, logs_move, (void
   *) (long)i);
5
6  /* Create pthreads for screen print and game status */
7  pthread_t control_threads[2]; // 0 for print, and 1 for status
8  pthread_create(&control_threads[0], NULL, screen_print, NULL);
9  pthread_create(&control_threads[1], NULL, status_judge, NULL);

```

The third part is to join the pthreads and output the result based on the game status. Whenever the game status changes to nonzero, all the pthreads created will exit.

```

1  /* join all the threads */
2  for (i = 1; i <= ROW - 1; i++)
3      pthread_join(wood_threads[i], NULL);
4  for (i = 0; i <= 1; i++)

```

```
5 pthread_join(control_threads[i], NULL);
6 /* Display the output for user: win, lose or quit. */
7 printf("\033[2J"); // clean the terminal
8 printf("\033[H"); // move the cursor to the top left corner
9 switch (game_status)
10 {
11 case 1:
12     printf("You win the game!!\n");
13     break;
14 case 2:
15     printf("You lose the game!!\n");
16     break;
17 case 3:
18     printf("You exit the game!!\n");
19     break;
20 default:
21     break;
22 }
23 /* exit the pthread */
24 pthread_exit(NULL);
25 return 0;
```

2.4 Implementing useful helping functions

The four useful helping functions are `int kbhit(void)`, `void map_print(void)`, `rand_num_gen(void)`, and `get_mod(int, int)`.

The `int kbhit(void)` function is used to detect whether there is a keyboard clicking. If yes, it will return 1, otherwise it will return 0.

The `void map_print(void)` function is used to print the map. It will firstly lock the mutex, and then print the map. After printing, it will unlock the mutex.

The `rand_num_gen(void)` function is used to generate a random number. It will use the `time()` function to get the current time, and then use the `srand()` function to set the seed. Finally, it will use the `rand()` function to generate a random number.

The `get_mod(int, int)` function is used to get the mod of two integers. It will firstly get the mod, and then judge whether it is negative. If yes, it will add the divisor to it. This function will be used frequently in the following codes.

The details of these codes are attached and are not be shown in the report.

2.5 Implementing the logs pthreads

The logs pthreads are used to control the logs. It will firstly get the current time, and then use the `rand_num_gen()` function to generate a random number. This random number will be used to initialize the places of each log. The logs will move to the right, or to the left based on the row value. Then it uses the function `void logs_shifting(int, int)` to move the logs.

The details are shown below.

```

1  /* shift the logs in map */
2  void logs_shifting(int row, int direc)
3  {
4      int left_end;
5      int right_end;
6      int center = get_mod(log_center[row], COLUMN - 1);
7      int half_log = LOG_LEN / 2;
8
9      switch (direc)
10     {
11     case 1: // left
12         left_end = get_mod(center - half_log - 1, COLUMN - 1);
13         right_end = get_mod(center + half_log, COLUMN - 1);
14         map[row][left_end] = '=';
15         map[row][right_end] = ' ';
16         if (is_frog_on_log(row, frog.x, frog.y))
17         {
18             map[row][frog.y] = '=';
19             if (frog.y == right_end)
20                 map[row][frog.y] = ' ';
21             frog.y = get_mod(frog.y - 1, COLUMN - 1);
22             map[row][frog.y] = '0';
23         }
24         log_center[row] = get_mod(center - 1, COLUMN - 1);
25         break;
26     case 0: // right
27         left_end = get_mod(center - half_log, COLUMN - 1);
28         right_end = get_mod(center + half_log + 1, COLUMN - 1);
29         map[row][left_end] = ' ';
30         map[row][right_end] = '=';
31         if (is_frog_on_log(row, frog.x, frog.y))
32         {
33             map[row][frog.y] = '=';
34             if (frog.y == left_end)
35                 map[row][frog.y] = ' ';
36             frog.y = get_mod(frog.y + 1, COLUMN - 1);
37             map[row][frog.y] = '0';
38         }
39         log_center[row] = get_mod(center + 1, COLUMN - 1);
40         break;

```

```

41     default:
42         break;
43     }
44 }

```

In the logs pthreads, if we click the keyboard, it will go to the process of moving frog. We should use kbhit to detect it and use the function `void frog_move_help(char ch)` to update the place of the frog.

In details, we should judge the index of the frog for different updating. The details are shown below.

```

1  /* help moving the frog */
2  void frog_move_help(char ch)
3  {
4      int old_x = frog.x;
5      int old_y = frog.y;
6      int half_log = LOG_LEN / 2;
7      int center = log_center[old_x];
8      /* process previous entry */
9      if (old_x == ROW || old_x == 0) // frog is on the bottom
10         bank
11         map[old_x][old_y] = '|';
12     else if (old_x != ROW && old_x != 0) // frog is on the river
13     {
14         if (is_frog_on_log(old_x, old_x, old_y))
15             map[old_x][old_y] = '=';
16         else
17             map[old_x][old_y] = ' ';
18     }
19     /* move the frog index */
20     switch (ch)
21     {
22     case 'W': case 'w':
23         if (frog.x != 0)
24             frog.x -= 1;
25         break;
26     case 'S': case 's':
27         if (frog.x != ROW)
28             frog.x += 1;
29         break;
30     case 'A': case 'a':
31         if (frog.y != 0)
32             frog.y -= 1;
33         break;
34     case 'D': case 'd':
35         if (frog.y != COLUMN - 2)
36             frog.y += 1;
37         break;
38     }
39 }

```

```

37     case 'Q': case 'q':
38         game_status = 3;
39         break;
40     default:
41         break;
42 }
43 /* set new frog on map */
44 map[frog.x][frog.y] = '0';
45 }

```

2.6 Implementing the control threads

The control threads include two parts, one for printing and another for game status judging.

For the printing, we should frequently refresh the terminal, i.e. print the latest map. This thread will exit when the game status changes.

For the game status, whenever the game reaches the condition to finish, it will change the game status and exit.

The details for the game status thread are shown below.

```

1  /* judge the status all the time */
2  void *status_judge(void *t)
3  {
4      while (1)
5      {
6          pthread_mutex_lock(&wood_lock);
7          /* Check game's status */
8          if (game_status != 0)
9          {
10             pthread_mutex_unlock(&wood_lock);
11             pthread_exit(NULL);
12         }
13         /* Check if the frog is on the top bank */
14         if (frog.x == 0)
15         {
16             game_status = 1; // win the game
17             pthread_mutex_unlock(&wood_lock);
18             pthread_exit(NULL);
19         }
20         /* Check if the frog touches the edge */
21         else if (frog.y == 0 || frog.y == COLUMN - 2)
22         {
23             game_status = 2; // lose the game
24             pthread_mutex_unlock(&wood_lock);
25             pthread_exit(NULL);
26         }

```



```

27     /* Check if the frog is on the river */
28     else if (frog.x != ROW && frog.x != 0)
29     {
30         if (!is_frog_on_log(frog.x, frog.x, frog.y))
31         {
32             game_status = 2; // lose the game
33             pthread_mutex_unlock(&wood_lock);
34             pthread_exit(NULL);
35         }
36     }
37     pthread_mutex_unlock(&wood_lock);
38 }
39 }

```

3 Executing and output

To execute the file, you can follow the command line below.

```

1 g++ hw2.cpp -lpthread
2 ./a.out

```

And then you will see the screen like this:

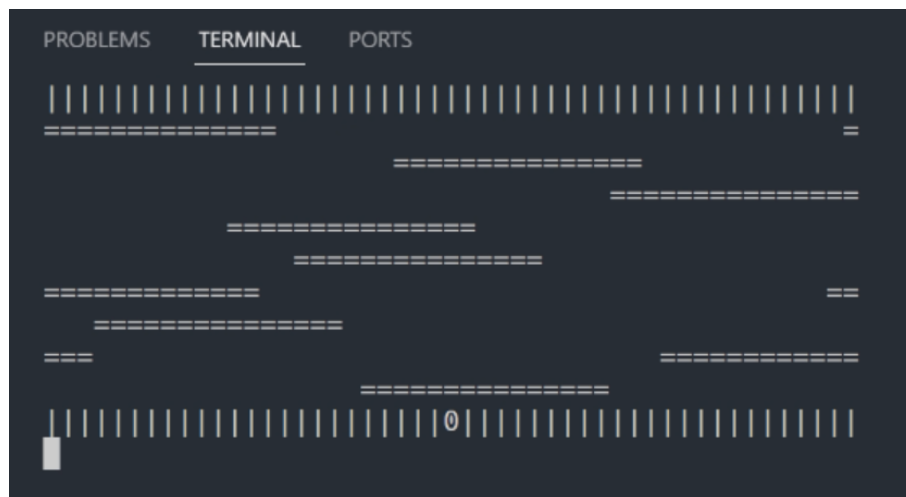


Figure 1: Starting screen

Then you can go up, down, left, right when you type "W", "S", "A", "D". You need to reach each logs step by step, shown in Figure 2, 3.

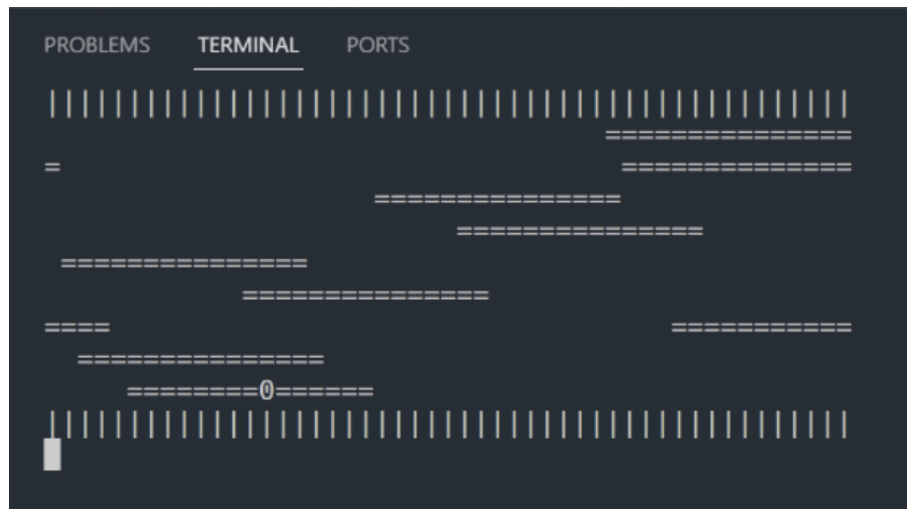


Figure 2: Moving the frog - 1

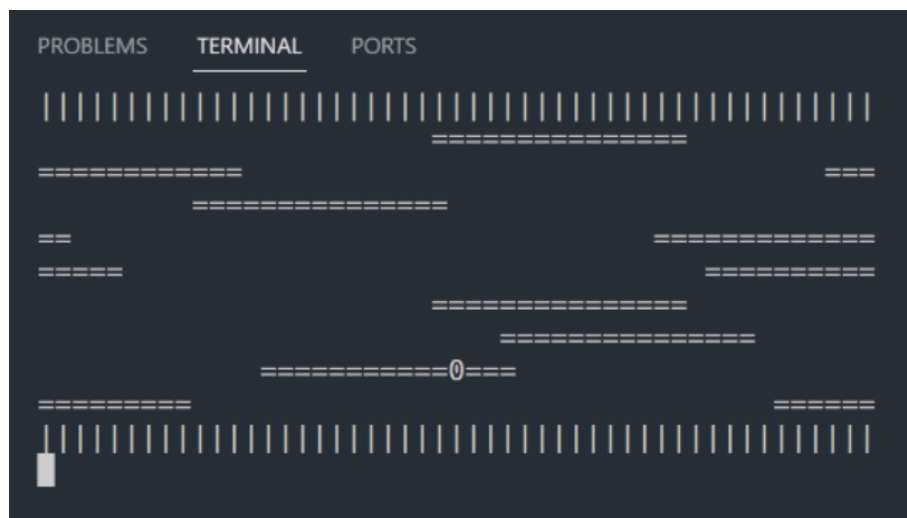


Figure 3: Moving the frog - 2

Continue to do so, If you successfully reach the top bank, you will win the game, shown in Figure 4, 5, 6.

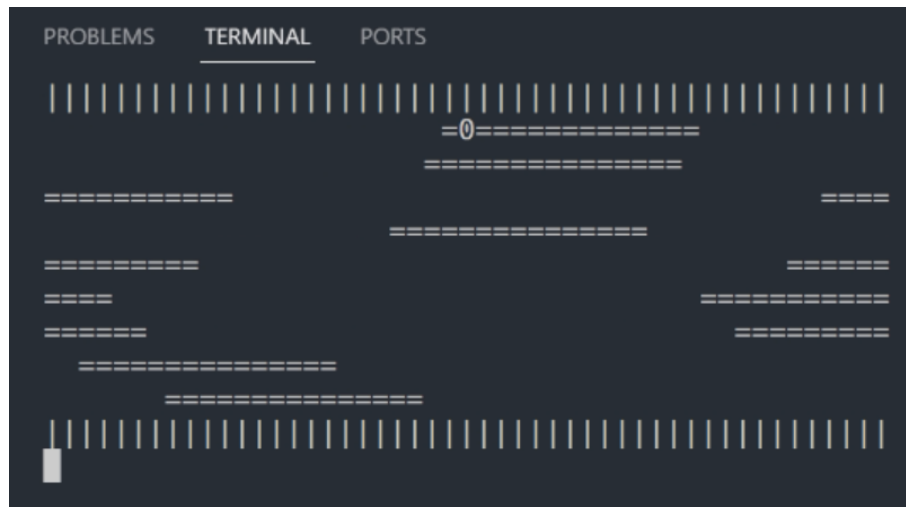


Figure 4: Win the game - 1

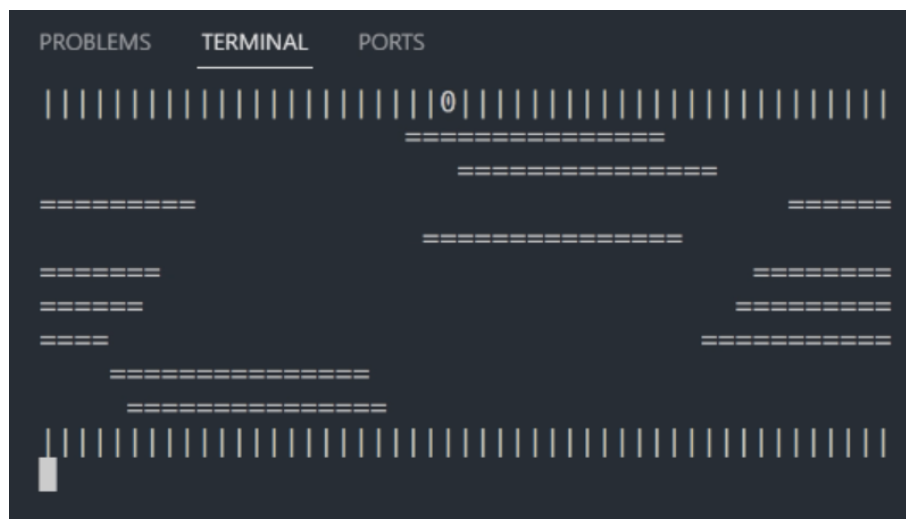


Figure 5: Win the game - 2

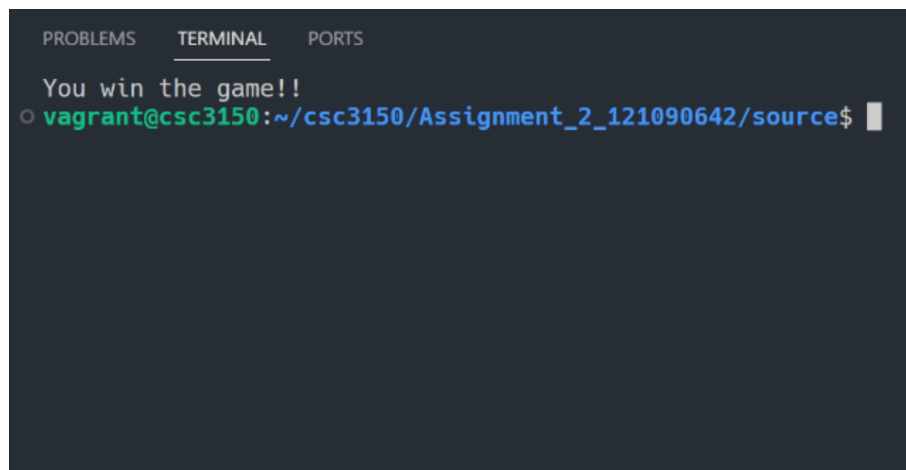


Figure 6: Win the game - 3

If you unfortunately touch the boundary or drop into the water, you will lose the game, as shown in Figure 7.

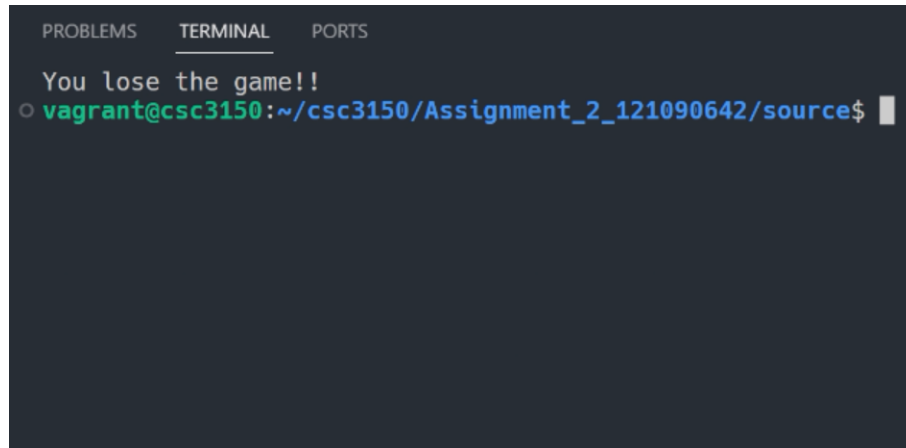
A screenshot of a terminal window with a dark background. At the top, there are three tabs: 'PROBLEMS', 'TERMINAL' (which is selected and underlined), and 'PORTS'. The terminal displays the text 'You lose the game!!' in white. Below this, there is a prompt character 'o' followed by the text 'vagrant@csc3150:~/csc3150/Assignment_2_121090642/source\$' in a light blue color. A white cursor is positioned at the end of the prompt line.

Figure 7: Lose the game

If you type "Q", you will quit the game, as shown in Figure 8.

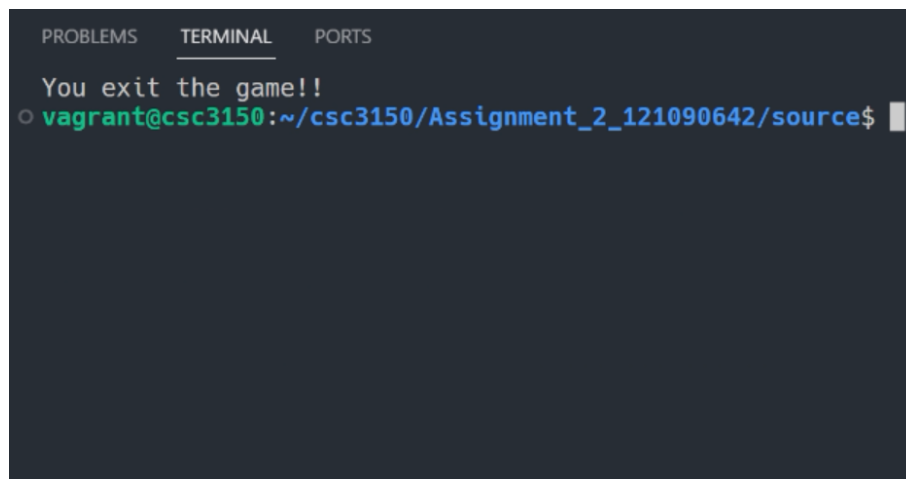
A screenshot of a terminal window with a dark background. At the top, there are three tabs: 'PROBLEMS', 'TERMINAL' (which is selected and underlined), and 'PORTS'. The terminal displays the text 'You exit the game!!' in white. Below this, there is a prompt character 'o' followed by the text 'vagrant@csc3150:~/csc3150/Assignment_2_121090642/source\$' in a light blue color. A white cursor is positioned at the end of the prompt line.

Figure 8: Exit the game

Here just shown some screenshots of important points. For the further understanding of the game, you can execute the game and play it by yourself.

4 Learn from the task

From the task, I have learnt something below:

1. How to compile a cpp file with multiple pthreads in Linux command line.
2. How to use `printf()` function to control the cursor on the terminal. For example, `printf("\033[2J")` is used to clean the terminal. This is important to create a mini game based on cpp on terminal.
3. How to create multiple pthreads. How to lock the mutex, and unlock them to protect the shared space. How to exit each pthread for safe termination.