Assignment Report: Report of 2nd Assignment of CSC3150

Meng Shuhan - 123090422

1 Introduction [2']

This assignment implements a multi-threaded terminal game named **"The Greatest Adventurer"**. In this game, an adventurer moves inside a dungeon to collect gold shards ($) while avoiding moving walls (=). The game runs entirely in the Linux terminal and is implemented using the C programming language with the POSIX Threads (Pthread) library.

The main objectives of this project are:

- To design a concurrent system using Pthreads where multiple entities move independently.
- To control game objects (player, walls, and golds) in real time using multi-thread synchronization.
- To manage user inputs (W/S/A/D/Q) asynchronously while ensuring smooth visual updates.

This project demonstrates a combination of concurrency control, real-time user interaction, and basic terminal graphics in C.

2 Design [5']

2.1 Overall Structure

The program is composed of three major components:

- **Main Thread:** Initializes the map, creates all worker threads.
- **Input Thread:** Listens to keyboard input (W/S/A/D/Q) using non-blocking input and updates the player's position.Check whether the player has touches the walls or gold.
- **Move Threads:** Six walls move alternately left and right in their respective rows. When a wall reaches one border, it reappears from the opposite side. Check whether walls touch the player. Six gold shards move in random directions. Check whether gold touches the player.

2.2 Synchronization

Because multiple threads modify the same shared resource — the `map[][]` array — a `pthread_mutex_t` lock is used to ensure mutual exclusion when updating positions or redrawing the map. This prevents screen flickering and race conditions.

2.3 Game Logic

- **Player Movement:** The player ('0') moves according to input keys: W (up), S (down), A (left), D (right). The adventurer cannot cross dungeon borders.
- **Collision Detection:** If the player moves into a wall ('='), the game prints `GAME OVER` and all threads terminate. If the player moves into a gold shard ('$'), that shard disappears. When all gold shards are collected, the game displays `YOU WIN!`.

- **Wall and Gold Movement:** Each wall and gold shard runs in own thread, sleeping for a short interval between moves to control speed.

3 Environment and Execution [2']

3.1 Environment

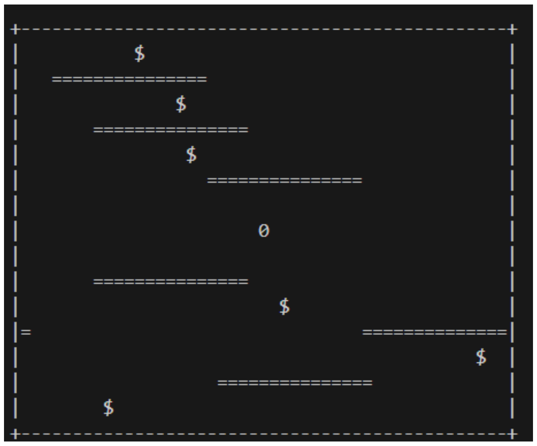
- Operating System: Ubuntu 16.04.7 LTS
- Linux Kernal Version: 5.15.10
- Compiler: g++ (Ubuntu 5.4.0-6ubuntu1 16.04.12) 5.4.0 20160609

3.2 Execution

```
$ g++ hw2.cpp -lpthread
$ ./a.out
```

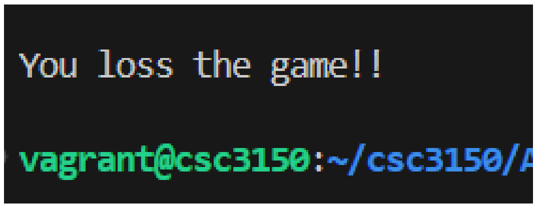
Once the program starts:

- Use `W/S/A/D` to move the adventurer.
- Press `Q` to quit at any time.



```
+-----------------------------------------------+
|           $                                   |
|   ===============                             |
|               $                               |
|       ===============                         |
|                $                              |
|                  ===============              |
|                                               |
|                       0                       |
|                                               |
|       ===============                         |
|                         $                     |
|=                                ==============|
|                                            $  |
|                   ===============             |
|        $                                      |
+-----------------------------------------------+
```

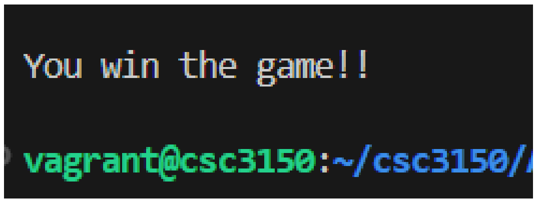
Figure 1: running status



```
You loss the game!!

vagrant@csc3150:~/csc3150/
```

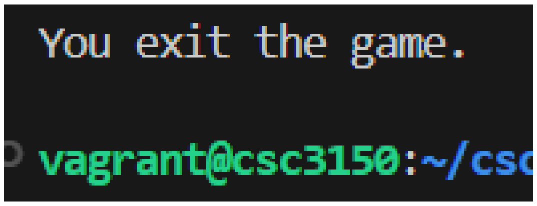
Figure 2: lose result



```
You win the game!!

vagrant@csc3150:~/csc3150/
```

Figure 3: win result



```
You exit the game.

vagrant@csc3150:~/csc
```

Figure 4: exit result

4 Conclusion [2’]

Through this assignment, I have learned how to use the Pthread library to design concurrent programs and manage synchronization between multiple threads safely. This project improved my understanding of:

- Mutex and thread safety in shared-memory environments.
- Real-time keyboard handling using non-blocking I/O.
- Designing smooth terminal-based animations with multi-thread coordination.

Overall, this assignment was a valuable experience in combining system-level programming concepts with interactive game design.