

---

# Assignment Report: The Greatest Adventurer Game

---

Chaoren WANG - 122090513

## 1 Introduction [2']

This assignment is about creating a game called “The Greatest Adventurer”. In this game, an adventurer moves in a dungeon to collect gold shards while avoiding moving walls. I worked on implementing the game using the C programming language and pthreads for multithreading. My tasks included setting up the game map, managing the movement of walls and gold shards, handling user inputs, and ensuring the game runs smoothly without errors.

## 2 Design [5']

I developed the program by first initializing a  $17 \times 49$  grid to represent the dungeon’s layout. The borders of the dungeon are created using specific characters: ‘+’ symbols mark the corners, ‘-’ characters form horizontal lines, and ‘|’ characters define the vertical borders. The adventurer, represented by ‘O’, is initially positioned at the center of the dungeon, at coordinates (8, 24).

The game contains six walls and six gold shards, each with distinct behaviors. Walls are represented by a string of 15 ‘=’ characters that move horizontally across the dungeon, alternating between left and right directions. Each wall has its own dedicated thread to manage movement and boundary conditions. If a wall reaches the end of the grid, it wraps around and reappears from the opposite side. Gold shards, represented by ‘\$’ symbols, move randomly and are also handled by separate threads, allowing independent and dynamic movement across the dungeon.

The game uses mutexes (`pthread_mutex_t`) to manage access to shared resources, such as the game map (`map_grid`) and the adventurer’s position, ensuring thread-safe operations in a multi-threaded environment. The primary thread is responsible for continuously printing the map to the screen, while other threads manage wall and gold shard movements, as well as handle user inputs that control the adventurer’s movement within the dungeon.

To initialize the game environment, the `initialize_map` function sets up the map layout and places the adventurer at the center. The `initialize_walls` function positions six walls at predetermined rows, with each wall having a random starting column and an initial movement direction, either left or right. Similarly, the `initialize_golds` function places six gold shards at specific rows with random starting columns. If a gold shard’s initial position is already occupied, the function searches for the next available spot.

In the main game loop, `move_walls_thread` continuously updates each wall’s position, moving it left or right. If the adventurer collides with a wall, the game ends in a loss. The `move_golds_thread` moves each gold shard randomly left or right. When the adventurer reaches a gold shard, it is collected, and the total count (`shards_remaining`) is decremented. Collecting all gold shards results in a win, indicated by an update in the `game_status_code`. A dedicated `input_thread_func` captures player movements (W/A/S/D for up, left, down, right) and can exit the game with the Q key. The `print_map_thread` continuously clears and prints the current map state to the terminal.

To manage terminal behavior, `setup_terminal` configures non-blocking input, while `reset_terminal` restores the default terminal settings upon game exit. After initialization, the program creates threads for moving walls, moving gold shards, printing the map, and handling input. Upon game completion, the main function waits for input and printing threads to conclude, then terminates the re-

maining threads, resets terminal settings, clears the screen, and displays the game result based on 'game\_status\_code'.

The game logic includes:

- Movement Controls:
  - W: Move Up
  - A: Move Left
  - S: Move Down
  - D: Move Right
  - Q: Quit the Game
- Objective:
  - Collect all six gold shards ('\$') scattered in the dungeon.
  - Avoid moving walls('='); colliding with a wall results in an immediate loss.
  - Collecting all gold shards ('\$') triggers a win.
- Display:
  - The adventurer, represented by '0', starts at the dungeon's center.
  - Walls move horizontally, alternating between directions.
  - Gold shards move randomly and can be collected by the adventurer.

### 3 Environment and Execution [2']

The program was developed on Ubuntu 22.04.3 LTS with a Linux kernel version 6.8.0-45-generic. And tested on Ubuntu 22.04.3 LTS with a Linux kernel version 5.10.1. The CC version used was 11.4.0, and GCC version used was 11.4.0. To compile the program, navigate to the source directory and use the following command:

```
gcc -pthread hw2.cpp -o hw2
```

Or use CC to compile:

```
cc -pthread hw2.cpp -o hw2
```

To run the game, execute:

```
./hw2
```

After executing it, the game running like Figure 1.

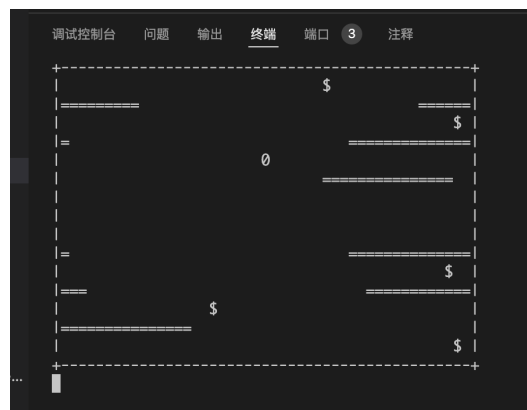


Figure 1: Game in action

In the game, use the keys W/A/S/D to move the adventurer up, left, down, and right respectively. Press Q to quit the game at any time. After collecting all the gold shards, the game will print "You

win the game!!”, as showed in Figure 2. Also, for exit and lose the game, there will be corresponding messages, like Figure 3 and 4.

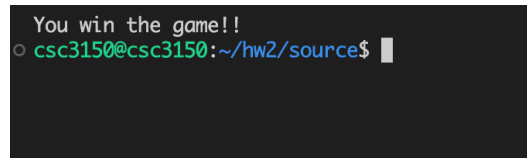
A terminal window with a dark background. The first line shows the message "You win the game!!" in white text. The second line shows a green prompt "csc3150@csc3150:~/hw2/source\$" followed by a white cursor.

Figure 2: Win the game

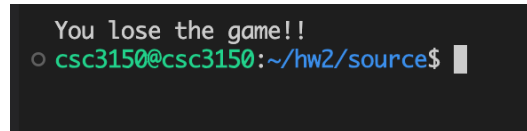
A terminal window with a dark background. The first line shows the message "You lose the game!!" in white text. The second line shows a green prompt "csc3150@csc3150:~/hw2/source\$" followed by a white cursor.

Figure 3: Game lose

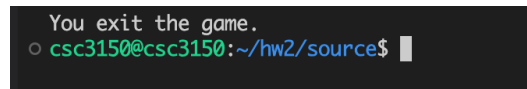
A terminal window with a dark background. The first line shows the message "You exit the game." in white text. The second line shows a green prompt "csc3150@csc3150:~/hw2/source\$" followed by a white cursor.

Figure 4: Game quit

## 4 Conclusion [2']

This assignment helped me understand how to implement multithreading in C using pthreads. I learned how to manage shared resources with mutexes to prevent conflicts between threads, handle real-time user inputs, and update the game map efficiently. Overall, I gained valuable experience in designing and developing a simple terminal-based game.