Herriot Watt University

School of Engineering and Physical Sciences

Electrical and Electronics Engineering Department

Robotics and Autonomous Systems Engineering

Praxis Game Project

Ravalitha Ravi Chander Poondamalli

H00473223

Dr. Girish Balasubramanian

13th April 2025

Praxis Game Project

1. Introduction

This report describes the design and implementation of a C-based console game titled "The Legend of the Lost Treasure of Blacktide Cove". The game simulates a treasure-hunting adventure across multiple grid-based levels, combining randomness, strategy, and power-ups. It offers increasing difficulty and an inventory system, providing replayability and player progression. The report includes code design details, game mechanics, flow diagrams, and user instructions.

2. Game Design and Mechanics

2.1. Game Objectives

The player's goal is to explore hidden grid spaces, avoid traps, collect treasures, and accumulate the highest possible score across three levels of increasing difficulty. The game ends when the player either finishes all levels or runs out of lives.

2.2. Core Features

Table 1: Features and Descriptions of the Game

Feature	Description
Grid-based Levels	Each level is a 2D grid with hidden objects (treasure, traps, keys, etc.).
Avatars	Players select one of 3 unique pirate avatars.
Inventory System	Tracks collected keys, maps, jewels, and score.
Power-ups	Includes extra guesses, maps to reveal traps, and jewels for bonus points.
Dynamic Difficulty	Players choose Easy, Medium, or Hard affecting traps, guesses, and value.
High Score System	Stores and displays high score across sessions using a file.

3. Game Flow and Logic

The game starts with a story to set the scene, then the player chooses their avatar and the difficulty level. Each level is a grid where different items like treasures, traps, and power-ups are hidden. The player has a limited number of guesses and lives to find the treasure. After each level, they get to pick a mystery chest that might help or hurt them. The game ends either when all levels are complete, or the player runs out of lives.

3.1. High-Level Flowchart

This diagram shows how the game flows from start to finish. It includes choosing an avatar, playing levels, handling guesses, and finishing the game.

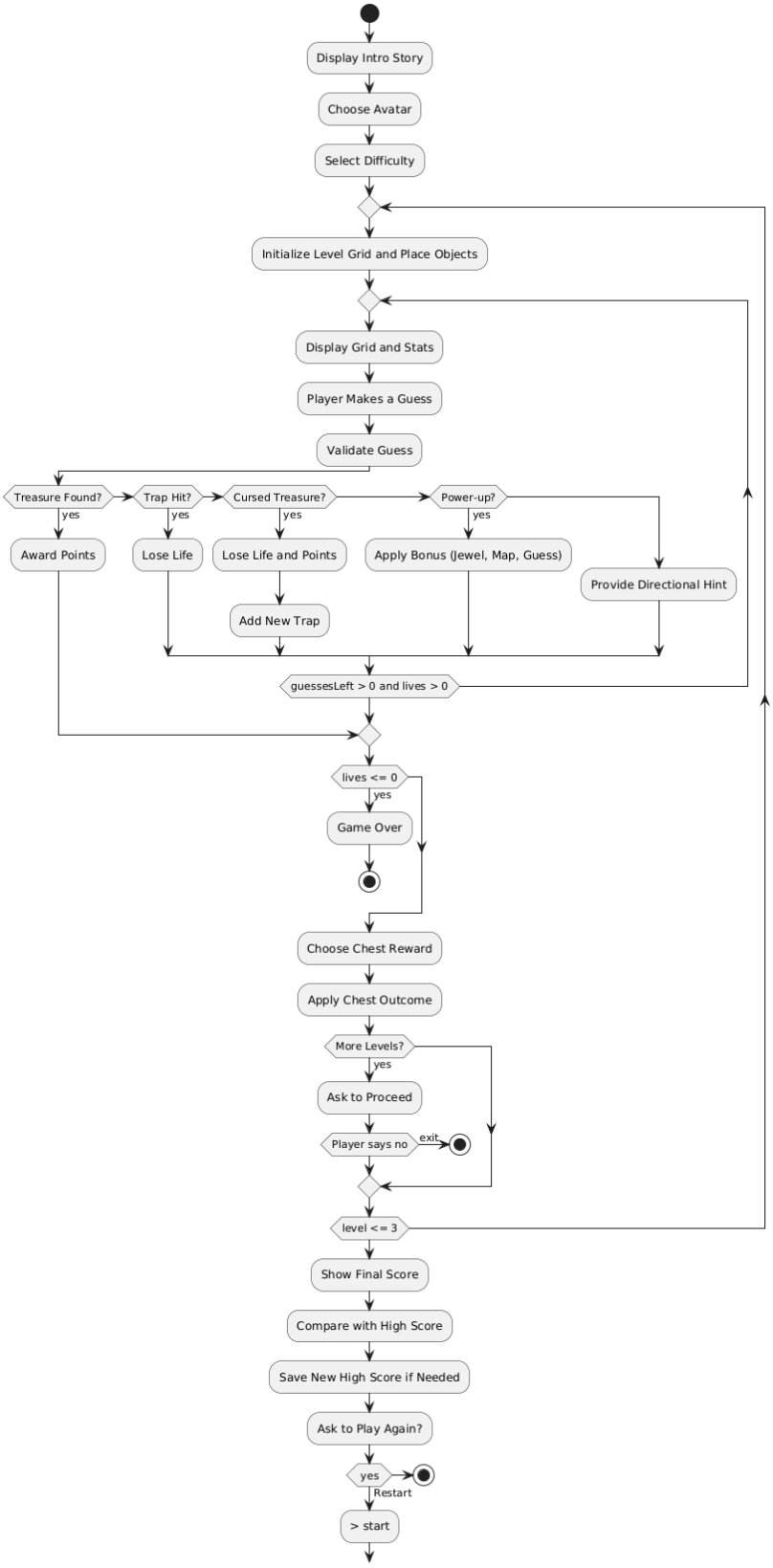

Figure 1: Game Flow

3.2. Object Symbols and Legend

On each turn, the player guesses a coordinate on the grid by entering a row and column number. The game checks what's hidden at that spot and responds depending on what's there.

Table 2: Symbols Explanation

What You Hit	What Happens
Treasure (`T`)	You find the treasure and get points. You finish the level.
Locked Treasure (`L`)	You need a key to open it. If you have a key, you unlock it and win.
Trap (`X`)	You lose a life. The trap is shown as `!` on the grid.
Cursed Treasure (`C`)	You lose a life and 5 points. A new trap is added somewhere randomly.
Key (`K`)	You add a key to your inventory. It can open a locked treasure.
Jewel (`J`)	You get bonus points (+20).
Extra Guess (`G`)	You get one extra guess added to your limit.
Map (`M`)	All traps (`X`) are shown on your grid as `!`, helping you avoid them.
Empty Space (`.`)	You get a hint about the direction of the treasure.

If you run out of guesses or lives, the level ends. If you find the treasure, you move on.

3.3. Levels and Difficulty

The game has 3 levels, and each level is a bit harder:

- The grid gets bigger each time (starts at 5×5, then 6×6, etc.).
- The number of traps depends on difficulty (easy = fewer traps, hard = more).
- Treasure is worth more on easier difficulties.

Table 3: Levels and Treasure Values

Difficulty	Guesses	Traps	Treasure Value Multiplier
Easy	8	2	15
Medium	6	3	10
Hard	4	4	5

3.4. Bonus Chests

After each level, the player picks a mystery chest (A, B, or C). It can contain:

- Bonus item (+20 points),
- Nothing (empty),
- Trap (lose 1 life).

3.5. Game End

The game ends in one of the below conditions and final score is compared with the saved high score.

- All levels are completed.
- You run out of lives.

4. Key Implementation Details

This section explains how the main parts of the game were built using C programming. It focuses on how the game handles the grid, how objects like treasures and traps are placed, how guessing works, and how lives and scores are managed.

4.1. Using Two Grids

The game uses two 2D arrays for the grid:

- grid – what the player sees (e.g. unexplored tiles, guessed spots, revealed treasures)
- hiddenGrid – stores all the hidden items like treasure, traps, keys, etc.

This setup keeps the hidden items secret while letting the game show the player only what they've discovered. Both grids are filled with . at the start, meaning unexplored space.

4.2. Placing Items Randomly

The function placeObjects() is used to randomly place everything on the hidden grid:

- One treasure (T)
- One locked treasure (L)
- One key (K)
- One cursed treasure (C)
- Three traps (X)
- Two random power-ups (J or G)
- One map (M)

This functions as follows:

- The code uses rand() to pick a random spot.
- A do...while loop is used to make sure the spot is empty before placing an item.
- This is then part is repeated for each object type.

4.3. Guessing and Checking Tiles

The main logic for what happens when the player guesses a tile is in the processGuess() function.

1. Subtract 1 from guesses left.
2. Check what's at the guessed location on the hiddenGrid.
3. Update the visible grid and change lives, score, or inventory depending on what was found.

Example from the code:

```
if (cell == 'T') {
grid[guessX][guessY] = 'T'; //reveal the treasure
return 1; //found treasure
} else if (cell == 'L') {
if (*keys > 0) {
printf(YELLOW "You unlocked the treasure! Claim yer gold!\n" RESET);
grid[guessX][guessY] = 'T';
(*keys)--; //use one key
return 1; //unlocked treasure
} else {
printf(RED "Locked treasure! Ye need a key to break it open!\n" RESET);
grid[guessX][guessY] = 'L';
return 3; //locked treasure
```

Figure 2

4.4. Managing Lives, Guesses, and Inventory

The game tracks several important player stats:

- lives: Starts at 3 and decreases when you hit a trap or cursed treasure.
- guessesLeft: Starts at 6 per level.
- score: Goes up when finding treasures and jewels, down when hitting cursed treasure.
- keys: Used to unlock locked treasures.

The stats are shown at the start of each turn like this:

```
GUESSES: 8 | LIVES: 3 | SCORE: 0
LOOT:
Gold: 0 | Gems: 0 | Maps: 0 | Keys: 0
```

Figure 3

4.5. Difficulty and Level Scaling

As the game moves through the levels

- The grid gets bigger each level: Level 1 = 5×5, Level 2 = 6×6, etc.
- Treasure value increases per level
- The number of guesses stays fixed (MAX_GUESSES = 6), but you can gain extras through power-ups.

4.6 Choosing Avatars and Chests

At the start, the player chooses from 3 pirate avatars using:

```
//let the player choose an avatar
int selectedAvatarIndex = selectAvatar(avatars, numAvatars);
```

Figure 4

After each level, the player picks a chest:

- Chest A = bonus points
- Chest B = nothing
- Chest C = trap (lose a life)

This adds surprise and randomness after every level.

4.6. High Score File System

The game keeps a high score saved in a file (high_score.txt) so you can try to beat your best score each time.

- loadHighScore() reads the file.
- saveHighScore() writes the new high score if you beat it.

```
//load the high score
int loadHighScore() {
    FILE *file = fopen(HIGH_SCORE_FILE, "r");
    if (file == NULL) {
        return 0; // default high score if file doesn't exist
    }
    int highScore;
    fscanf(file, "%d", &highScore);
    fclose(file);
    return highScore;
}

//save the high score
void saveHighScore(int highScore) {
    FILE *file = fopen(HIGH_SCORE_FILE, "w");
    if (file == NULL) {
        printf("Error saving high score.\n");
        return;
    }
    fprintf(file, "%d", highScore);
    fclose(file);
}
```

Figure 5

5. Conclusion

This game project successfully combines elements of procedural generation, user interaction, file I/O, and grid-based logic into a well-working and fun terminal experience. By implementing dynamic scaling and modular code design, I ensured the game remains engaging across multiple playthroughs.