Treasure Hunt Game – Technical Report

Student: Mohammad

Module: COM4103 Computing Skills Portfolio

Institution: Leeds Trinity University

Date: May 2025

1. Introduction

This report describes the design and implementation of a Python based command line game called Treasure Hunt. This project was created as part of the COM4103 Computing Skills Portfolio module, and demonstrates the application of core Python programming concepts, algorithmic thought, and software development tools. The aim of the project was to use practical coding skills to create an interactive artefact incorporating player interaction, health related mechanics, and classical search algorithms. The artefact was made for two players, and included traps, power ups, and a hidden treasure within the gameplay. The winner was decided by the first player to find the hidden treasure.

2. Tools and Technologies Used

The following technologies were used to develop the project:

• Python 3.12: This core programming language was used to create the game logic and implementation.

• Visual Studio Code (VS Code): The IDE which has good support for version control, as well as being straightforward to set up and use.

• macOS Terminal: The app that was used to run and test the program and to set up gameplay from the command line.

• GitHub: Were used for version control and online submission

• ZIP file packaging: To package and submit the standalone artefact.

Python was selected as the programming language, for its readability, implementation ease when rapid prototyping, and availability of built in software libraries supporting

implementation of algorithmic logic (Python Software Foundation, 2023).

3. Project Overview

The game is played on a 5 x 5 grid, where each player will start in a random position. The grid is populated with various items:

- T: Treasure (1 total, randomly hidden in a cell)
- X: Traps (decrease player health)
- O: Obstacles (block movement)
- P: Power ups (restore health or give a hint)

Players will take turns to move around the grid using directional commands (up, down, left, right) or the algorithmic searches (bs, bfs, dfs). Players start at 5 health and lose health by stepping on the traps. If a player's health reaches zero, they lose. The first player to reach the treasure wins.

As well, the power ups for health and hints give the players a strategical advantage. The health power up would increase the player's health up to a max of 7. The hint power up will show a row or column where the treasure is located. The game keeps a turn based game loop. There are also collision detection, win/lose conditions, and feedback given to the user in the terminal when they play.

4. Algorithm Implementations

The unique aspect of this game is that we are able to use three of the classic search algorithms:

4.1 Binary Search (BS)

Binary search is used to emulate the lookup of whether a row or column has a power up or treasure. While binary search is used in sorted or traced arrays, we implemented it in the game's context, as a recursive scanning function, to pioneer looking for power ups and treasures in an unordered section of the grid. The player has the means to type in commands like "bs row 3".

4.2 Breadth First Search (BFS)

Breadth first search was applied to find the shortest route to the treasure from the player's current position. This is valuable when players want the fastest route to accomplish their goal. The algorithm uses a queue data structure to try every direction and avoid obstacles.

4.3 Depth First Search (DFS)

Depth First Search (DFS) is another pathfinding demonstration and exploration option for players that explores deep paths before backtracking. Players can utilize DFS to find paths in more detailed and complex grids. BFS and DFS both search for the path to the treasure, while avoiding obstacles and traps.

5. Testing and Evaluation

The DFS algorithm uses recursion to search deeply down one branch of movement before backtracking. DFS can be useful in cases where players want to find longer, or hidden paths around obstacles. Although a deep path is not always the shortest, it offers a different kind of pathfinding flexibility in the exploration of the grid when BFS paths may be blocked. The end of the implementation simply avoids going back to previously explored cells, so if the player reaches the treasure the game ends.

The game was tested using the command line interface manually. Some of the important cases tested were:

• valid and invalid player movement

• crashing into obstacles and traps

• collecting a power up (health and hint)

• using the search commands (bs, bfs, dfs)

• boundary checking, and not allowing movement outside of the grid

• winning the game and taking a player out of the game

No major bugs were encountered. Minor tweaks included enforcing a maximum health limit and improving user input validation.

In terms of performance, the game is efficient due to the small grid size and simple

data structures (lists, sets, and queues). The search algorithms completed instantaneously. Additional test cases involved surrounding a player with obstacles to test blocking and placing traps near spawn points to confirm health loss and elimination logic. Search results were visually verified via terminal output.

6. Experiences and Solutions

There were a number of experiences encountered during development:

• Python 3 compatibility: The Mac system originally ran Python 2.7. This was resolved by manually installing Python 3.12 and setting it as the default interpreter.

• Grid randomness: Random selection sometimes positioned items making it possible for the player to spawn next to a trap. This was resolved by restricting spawning in empty cells.

• Player collision: Logic checks were added to ensure a player can't move into the same cell as the other player.

Other refinements included inserting docstrings and comments for support, and an input system that unified movement and search into one turn. Another experience involved ensuring the input from the user to the game was consistent. The game was developed to ignore upper case letters and leading/trailing spaces, improving usability. Clear turn instructions were added to guide the players.

7. Ethical and Legal Considerations

The game does not store any personal data and does not utilize any online services. All gameplay occurs locally on the user's machine, and there is no tracking or analytics implemented. The game does not contain any copyrighted material.

The directions followed were those of open source applications and documentation, with references cited. The project was developed solely for educational purposes.

The game is designed to be fair: item placement is random but balanced for all players. No arbitrary advantages are granted. As a turn based game, each player has equal opportunity to strategize. Although it uses a CLI, future enhancements could include

voice commands or visual accessibility features.

8. Submission Details

This artefact has been submitted in the following form:

• ZIP file: Contains a copy of the python script treasure_hunt.py and README.md file.

• GitHub Repository: Publicly available at https://github.com/samee123x/com4103-treasurehunt.git

• Technical Report: This PDF file

All components reflect the final working version of the game as tested on macOS using Python 3.12.

9. References (APA 7)

• GitHub. (n.d.). "GitHub: Where the world builds software". https://github.com

• Python Software Foundation. (2023). "Python 3.12 documentation". https://docs.python.org/3.12/

• Stack Overflow. (n.d.). "How to implement BFS and DFS in Python?" https://stackoverflow.com/questions/8922060/how-to-trace-the-path-in-a-breadth-first-search