Treasure Hunt Game – Technical Report

Student: Mohammad Samee

Module: COM4103 Computing Skills Portfolio

1. Introduction

This report will outline the design and development of a Python game titled “Treasure Hunt”. As part of the COM4103 Software Development module, this project demonstrates key programming concepts like algorithmic thinking, modular design & interactive gameplay mechanics. The purpose of the game is to produce a functional and engaging two-player game which includes features like traps, power-ups & hidden treasure within the game. First player to find the hidden treasure secures the win.

2. Tools and Technologies Used

The following tools 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 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)

Using directional commands such as up, down, left & right or algorithmic searches (bs, bfs, dfs). Players will use these to move around the grid, the starting health point will be 5 and each time a player steps on a trap the player loses its health. Once they reach zero, they lose. The first player who reaches the treasure wins. The power ups will increase the player’s health up to a maximum of 7. The hint power up will show a row or column where the treasure is located. The game will stay as a turn based game loop. The collision detection, win/lose conditions and feedback given to the user is within 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 .For 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 from its users. 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.

8. Submission Details

- 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

9. References (APA 7)

GitHub. (2025). *GitHub*. GitHub. https://github.com/

StackOverflow. (2022). *python - How to trace the path in a Breadth-First Search?* Stack Overflow. https://stackoverflow.com/questions/8922060/how-to-trace-the-path-in-a-breadth-first-search

Python.org. (2025). *3.12.1 Documentation*. Docs.python.org. https://docs.python.org/3.12/