TREASURE HUNT GAME

Introduction

The Treasure Hunt game is an interactive program designed to simulate a treasure-seeking adventure within a grid-based environment. It involves strategic movement, resource management, and a mix of random challenges. The game is coded in Python and utilizes classes and various built-in modules such as random and collections. Deque to create dynamic and engaging gameplay. The Treasure hunt game allows two people to play. One person who will be playing will be a real human and the other player will be AI opponent.

Project overview

The main goal of the Treasure Hunt game is to navigate a grid, avoid traps and obstacles, collect power-ups, and ultimately find the hidden treasure before a rival player does. The game initializes with a grid where elements such as traps (T), obstacles (O), power-ups (P), and the treasure (X) are randomly placed. The player and a rival start from opposite corners of the grid and make moves in turn, with the player aiming to reach the treasure first.

Game components

Grid Initialization: The game board is a grid of a specified size (default is 10x10). The grid is populated with traps, obstacles, power-ups, and a treasure.

Player and Rival: The player starts at the top-left corner of the grid, and the rival starts at the bottom-right corner. Both characters move across the grid seeking the treasure.

Health and Power-ups: The player's health starts at 100 and can be affected by encountering traps (-20 health) or collecting power-ups (+10 health). The player can collect multiple power-ups throughout the game.

Traps The two player will face traps while they move across the gird to collect the treasure. I have include the traps in the treasure hut game as it make the game more harder and interesting. Every time a player step or interact with a trap they will lose 20 health once the player hit zero health the game will be finish as the player will not have any health left to continue playing the game.

Challenges and solutions

One of the main challenges in developing the Treasure Hunt game is ensuring that the grid's random initialization remains fair and provides a balanced gameplay experience. The grid is filled with traps, obstacles, and power-ups using Python's random module, which ensures unpredictability in each game session.

Solutions

By implementing a balanced distribution algorithm within the initialize grid method, the game maintains a consistent level of difficulty. Traps and obstacles are randomly placed, but their quantities are controlled to prevent overly challenging or trivially easy gameplay.

TREASURE HUNT GAME

Pathfinding and rival movements

Another challenge is implementing an effective rival movement algorithm that simulates intelligent behaviour. The rival should seek the treasure but also navigate the grid efficiently, avoiding obstacles and traps.

Solution

The Breadth-First Search (BFS) algorithm is used to determine the shortest path for the rival to reach the treasure. This ensures that the rival moves logically and provides a competitive aspect to the game. The BFS implementation checks for obstacles and avoids revisiting cells.

Seach algorithms

The game's pathfinding mechanism is crucial for both rival movement and enhancing the competitive element. BFS is used to navigate the grid efficiently.

Breadth Depth Search

BFS is a fundamental algorithm in computer science used for traversing or searching tree or graph data structures

Solution

The bfs_pathfinding method is employed to determine the shortest path from the rival's current position to the treasure. This ensures the rival moves towards the goal efficiently and competes effectively against the player.

Pseudocode flow chart

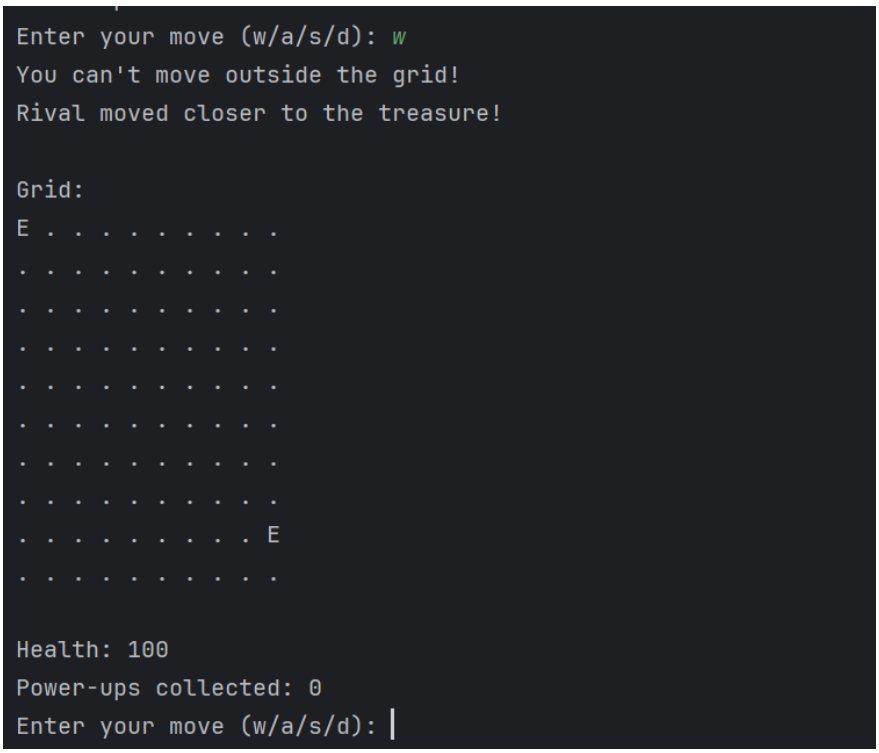
To better understand the flow of the game, here's a simplified pseudocode representation of the main components:

- Initialize Grid
- Place Traps, Obstacles, Power-ups, and Treasure
- Set Player and Rival Starting Positions
- While Player Health > 0:
- Display Grid
- Get Player Move
- Move Player
- Interact with Cell (Traps, Power-ups, Treasure)
- Move Rival (using BFS)
- Check Rival's Position
- If Rival Finds Treasure:
- End Game with Rival Win
- If Player Health <= 0:

TREASURE HUNT GAME

Test Screen Shots

As you can see from the screen shot below I have include so evidence that my game meet the requirement that I have been given I have a health bar power up movement and binary search.



TREASURE HUNT GAME

```
Health: 100
Power-ups collected: 0
Enter your move (w/a/s/d): f
Invalid move! Use 'w', 'a', 's', 'd' to move.

Grid:
E . . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . E .
. . . . . .

Health: 100
Power-ups collected: 0
Enter your move (w/a/s/d):
```

```
Health: 100
Power-ups collected: 0
Enter your move (w/a/s/d): s
You stepped on a trap! Health -20.

Grid:
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
E . . . . . E

Health: 80
Power-ups collected: 0
Enter your move (w/a/s/d):
```

TREASURE HUNT GAME

```
Health: 100
Power-ups collected: 0
Enter your move (w/a/s/d): s
You found a power-up! Health +10.
Rival moved closer to the treasure!

Grid:
. . . . . .
E . . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . E
. . . . . .
. . . . . .

Health: 110
Power-ups collected: 1
Enter your move (w/a/s/d): d
```

TREASURE HUNT GAME

```
Grid:
. . E . . . . . .
. . . . . . . . .
. . . . . . . . .
. . . . . . . . .
. . . . . . . . .
. . E . . . . . .
. . . . . . . . .
. . . . . . . . .
. . . . . . . . .
. . . . . . . . .

Health: 100
Power-ups collected: 0
Enter your move (w/a/s/d): a
Rival moved closer to the treasure!
The rival found the treasure before you! Game over.

Process finished with exit code 0
```

TREASURE HUNT GAME

Flow chart description

- **Initialize Grid**: The grid is created and populated with game elements.
- **Player Turn**: The player makes a move, and the game updates the player's position.
- **Interact with Cell**: Depending on the cell content (trap, power-up, or treasure), the player's stats are updated.
- **Rival Turn**: The rival moves towards the treasure using BFS.
- **Check Positions**: The game checks if the rival has reached the treasure.
- **Game Over Conditions**: The game ends if the player's health drops to zero or the rival finds the treasure.

Testing evidences

Thorough testing is conducted to ensure the game functions as expected. The following scenarios are tested:

- **Grid Initialization**: Ensuring the random placement of traps, obstacles, power-ups, and treasure.
- **Player Movement**: Verifying the player can move correctly within the grid boundaries.
- **Rival Movement**: Checking the BFS pathfinding for accurate rival movements.
- **Health Updates**: Validating health changes upon interaction with traps and power-ups.
- **Game End Conditions**: Confirming the game ends correctly when the player's health depletes or the rival finds the treasure.

Conclusion

The Treasure Hunt game offers a captivating mix of strategy, luck, and competition. By leveraging Python's OOP principles, randomness for dynamic gameplay, and BFS for intelligent rival movements, the game provides an engaging experience for players. The implementation demonstrates key programming concepts and showcases the importance of balancing randomness with structured algorithms to create a fair yet challenging game environment.

https://github.com/virajleedstrinity/software-development-assignment-2.git