DSA Miniproject Console-based Monster-Rogue Chase game on an N × N dungeon

Build a TUI grid game on an N × N dungeon where a monster chases a rogue, and the rogue tries to avoid capture as long as possible. The dungeon contains walls, rooms, and corridors, and both characters move turn by turn. The monster uses a strategy to move one step closer to the rogue using shortest-path search (BFS/DFS), while the rogue moves to the adjacent square that maximizes distance from the monster. The game ends when the monster reaches the rogue.

Team:

PES2UG24CS042: Akash Singh

PES2UG24CS003: A Sheiman Joshi

PES2UG24CS019: Abhigyan Dutta

Project Overview & Key Features

- TUI-based N × N dungeon grid game, where a monster chases a rogue, and the rogue tries to avoid capture as long as possible
- The Dungeon contains walls, hidden rooms (the form of maze corners), and corridors, and both characters move turn by turn
- The monster uses a strategy to move one step closer to the rogue using shortest-path while the rogue moves to the adjacent square that maximizes distance from the monster.
- The game ends when the monster reaches the rogue.
- The player pressed enter to play the chances for this game.

Data Structures in this project

- Linear Data Structure-

  Queue(Linked List Representation)-Used for BFS traversal for getting shortest path, Implements FIFO (First In First Out), Operations: enqueue(), dequeue(), isEmpty()

- Non-Linear Data Structure-

  Graph-The N×N dungeon grid represents a graph, Each cell is a node, Adjacent walkable cells are edges, BFS explores this graph structure in the shortest path possible.

Why did we use those Data Structures :

Queue (Linear): Used to power the BFS (Breadth-First Search) algorithm.
Its FIFO (First-In, First-Out) nature is essential for exploring the graph layer-by-layer, which is the only way to guarantee finding the absolute shortest path for the monster.

Graph (Non-Linear): The N x N dungeon grid is the graph. It's used to represent t he map, its rooms, and its walls.
Nodes = Walkable cells
Edges = Paths between adjacent cells

Workflow

- The game displays a 15×15 dungeon grid

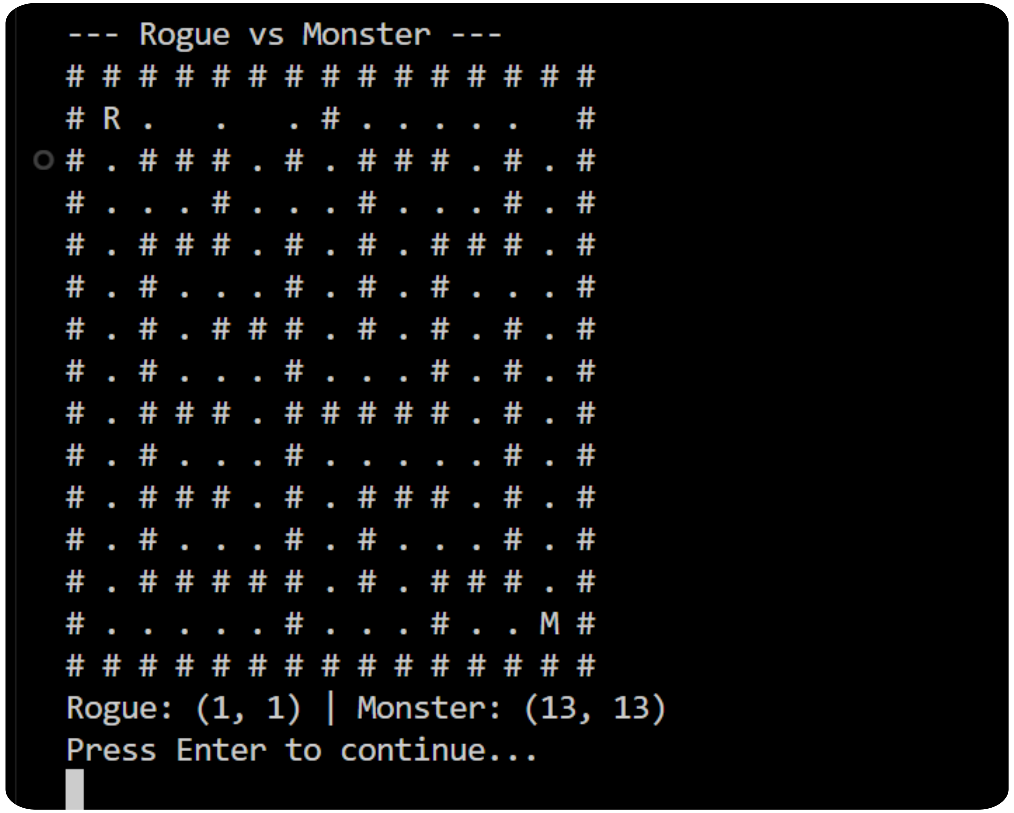
R=Rogue (you)

M= Monster (chaser)

#= Wall

. = Path

- Press Enter to advance each turn
- Rogue moves first(tries to run away)
- Monster moves second(chases using shortest path)
- Game continues until Monster catches Rogue



```
--- Rogue vs Monster ---
# # # # # # # # # # # # # # #
# R .   .   . # . . . . .   #
# . # # # . # . # # # . # . #
# . . . # . . . # . . . # . #
# . # # # . # . # . # # # . #
# . # . . . # . # . # . . . #
# . # . # # # . # . # . # . #
# . # . . . # . . . # . # . #
# . # # # . # # # # # . # . #
# . # . . . # . . . . . # . #
# . # # # . # . # # # . # . #
# . # . . . # . # . . . # . #
# . # # # # # . # . # # # . #
# . . . . . # . . . # . . M #
# # # # # # # # # # # # # # #
Rogue: (1, 1) | Monster: (13, 13)
Press Enter to continue...
```

Summary

- A Total of 12 Functions are included in this miniproject, namely: `createQueue(), enqueue(Queue* q, Point p), dequeue(Queue* q), isQueueEmpty(Queue* q), freeQueue(Queue* q), initGame(GameState* game), clearScreen()` (this is a win32 functionality which is optionally put by me to ensure the terminal isn't flooded), `printGame(GameState* game), isValid(GameState* game, int r, int c), moveRogue(GameState* game), moveMonster(GameState* game),` and finally, `main().`

Summary

- This project shows use of-

  Queue used for BFS traversal the monster uses BFS (shortest-path search) to chase.

- N x N dungeon grid represents a graph where each cell is node
- Adjacent cell represents edges
- The system effectively models monster-rogue chase game on N x N on a TUI (Terminal UI) – Based Interface.

Thank
You!