MAZE RUNNER

Supervisor:
Eng:Omar Khalled
Eng:Ahmed Sobhy

1. Problem Description

This project addresses a maze pathfinding problem where a rescue agent must navigate a haunted maze to reach a lost child while avoiding AI-controlled monsters.

The environment is non-static, meaning:

- Maze walls can appear or disappear during execution
- Monsters move continuously and have a detection radius
- The agent must replan its path whenever the environment changes

The goal is to find a safe and efficient path to the target under dynamic constraints.

2. Why This Problem?

This problem is an advanced extension of the classical Maze Pathfinding problem.

It is suitable because it:

- Maps naturally to a state-space search problem
- Supports the implementation and comparison of multiple search strategies, including BFS, DFS, Uniform Cost Search, A, and Hill Climbing*
- Demonstrates the limitations of local search in dynamic or constrained scenarios
- Provides flexibility and opportunities for creativity, in line with course objectives

Compared to a static maze, this problem is more challenging and realistic.

3. State Representation

Each state is represented as:

(x, y)

Where:

- x, y are the agent's current position in the maze.

The environment state also includes:

- Current maze layout
- Positions of all monsters

4. Successor Function

From any state (x, y), the agent can move to:

- Up
- Down
- Left
- Right

A move is valid if:

- The target cell is not a wall
- The cell is not inside any monster's detection radius

5. Cost Function

Cell Type	Cost
Free cell (0)	1
Monster (M)	5
Wall (1)	Not allowed

7. Dynamic Replanning Strategy

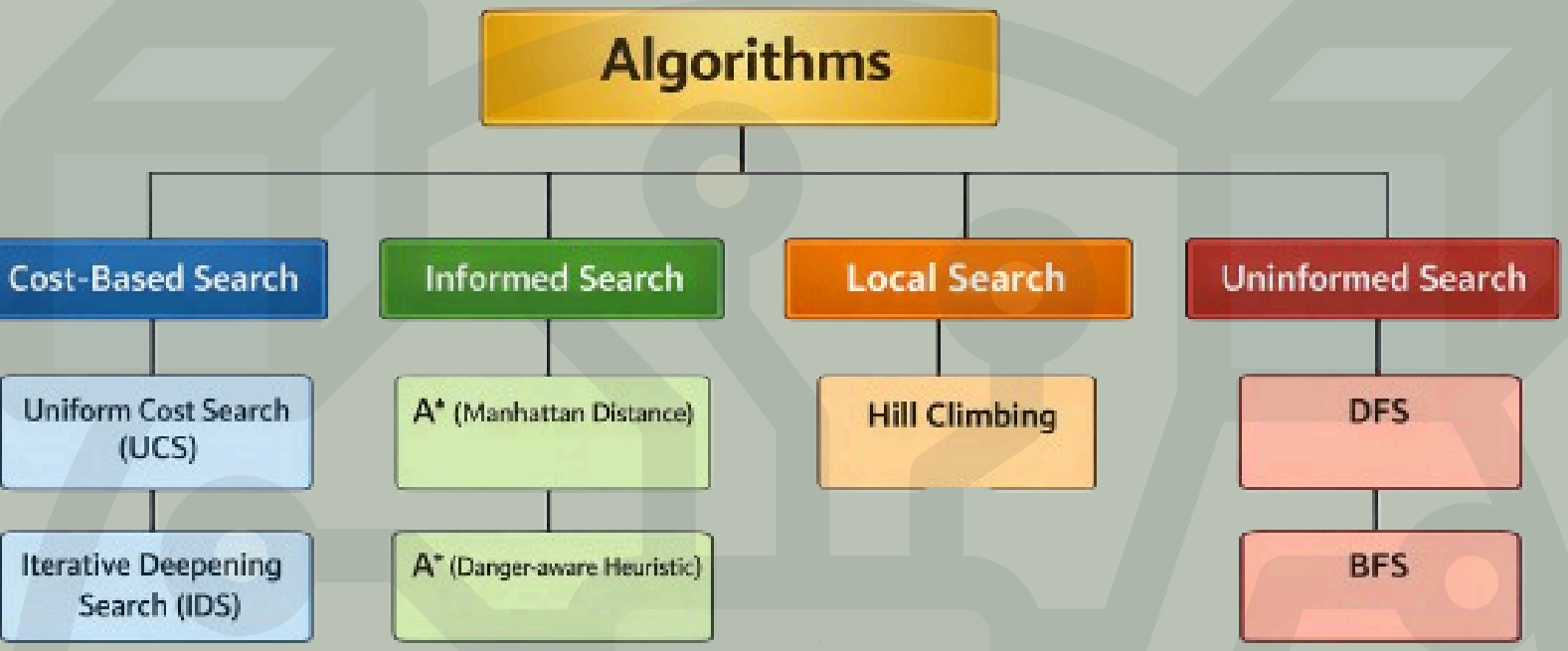
Since the environment is dynamic:

- **The agent continuously monitors:**
  - Monster movements
  - Maze wall changes
- **If the current path becomes unsafe or blocked:**
  - A* is executed again from the agent's current position

This demonstrates online search and replanning.

Simulation of the Maze Environment

Algorithm



maze Link

https://maze-runner-five-ruddy.vercel.app/

Uninformed Search

DFS (Depth-First Search)

DFS is a search algorithm that explores one path as deeply as possible before backtracking. It uses a stack or recursion.

How it works in our maze:

- Start from the rescue team's position
- Follow one path deeply
- If blocked or dangerous, backtrack and try another path
- Repeat until the child is found
-

Why DFS Can Be Good

- Low memory usage: stores only the current path
- Easy to implement: simple and fast for a prototype
- Finds a path if one exists (not necessarily the shortest)
- Good for full exploration of the maze
-

Why DFS Is Bad for Our Project

- Not optimal: does not find the shortest or safest path
- Ignores dangers: may go directly into monster zones
- Poor with dynamic environments: needs to restart often
- Can loop forever without proper visited checks
- No heuristic guidance: completely blind search
-

DFS vs A*

- A* uses cost + heuristic to find the best path efficiently
- DFS ignores both
- For a dynamic rescue game, A* (or D* Lite) is the better choice, while DFS is only useful for exploration

Uninformed Search

Breadth-First Search (BFS)

Maze Pathfinding is about finding the shortest path from a start point to a goal in a grid with obstacles. Some cells are free to move, and some are blocked. The goal is to reach the target quickly while avoiding walls.

BFS works level by level using a queue. It is good for this problem because all moves have the same cost. BFS always finds the shortest path if it exists. It is complete and optimal, but it can use a lot of memory for large mazes.

In conclusion, BFS is a simple and reliable method for Maze Pathfinding. It finds the shortest path and is easy to understand, making it a useful search strategy in AI.

Why BFS can be good?

Finds the shortest path guaranteed

Simple and easy to understand

Works well for small to medium mazes

Why BFS bad for our project?

Uses a lot of memory for large mazes

Slow for big mazes

Cost-Based Search

Uniform Cost Search (UCS)

UCS always expands the lowest total cost path first, ensuring the optimal solution.

Why UCS?

- Maze has different costs
- Safest and cheapest path is required
- Guarantees an optimal solution

lterative Deepening Search (lDS)

IDS combines:

- Depth-First Search memory efficiency
- Breadth-First Search completeness

It repeatedly applies Depth-Limited Search with increasing depth limits.

Why IDS?

- Maze size may change
- Limited memory
- Ensures solution without storing large trees

Informed Search

A* Search (Manhattan Distance)

A* is an informed search algorithm that uses both:
The actual cost from the start g(n)
A heuristic estimate to the goal h(n)

It evaluates nodes using:

$$f(n)=g(n)+h(n)$$

Heuristic Used:
Manhattan Distance:

$$h(n)= | x-xg | + | y-yg |$$

Why A*?

- Faster than UCS
- Reduces unnecessary node expansions
- Guarantees optimal solution when heuristic is admissible
- Widely used in games and pathfinding problems

Informed Search

A* (Danger-Aware Heuristic)

Maze Pathfinding is about finding the shortest path from a start point to a goal in a grid with obstacles. Some cells are free to move, some are blocked, and some may be dangerous or harder to cross. The goal is to reach the target quickly while avoiding walls and dangerous areas.

A* uses a priority queue and a heuristic function to guide the search. The Danger-Aware heuristic estimates the cost to the goal while considering dangerous cells as higher cost. This helps the algorithm find safer and shorter paths efficiently.

Why A* (Danger-Aware Heuristic) can be good?

Finds the shortest path efficiently

Can avoid dangerous areas using the heuristic

Why A* (Danger-Aware Heuristic) bad for our project?

May take a longer path to avoid danger

Higher total cost

Depends on monster or obstacle positions

Hill Climbing Algorithm

1. Overview

Hill Climbing is a local search algorithm that attempts to find an optimal solution by iteratively improving the current state based on a heuristic function.

Unlike global search algorithms (such as BFS or A*), Hill Climbing:

- Considers only the current state and its neighbors
- Does not store a search tree
- Moves greedily toward states with better heuristic values

When applied to a dynamic haunted maze, Hill Climbing is unable to respond effectively to environmental changes such as:

- Moving monsters with detection ranges

As a purely local search method, the algorithm often becomes trapped in local minima or follows unsafe paths. This limitation highlights the necessity of global search algorithms with replanning capabilities, such as A*, in dynamic environments.

In this project, Hill Climbing is applied to a snapshot of the dynamic haunted maze, without dynamic replanning, in order to evaluate its behavior and limitations within a changing environment.

Expected Results

A 15×21 grid maze with approximately 30% obstacles and 3 monsters

Algorithm	Path Length (steps)	Success Rate	Nodes Explored	Time (ms)
BFS	68	100%	4200	260
DFS	112	65%	2100	180
UCS	68	100%	3900	310
A* (Manhattan)	68	100%	1350	95
A* (Danger-aware)	74	100%	1650	130
Hill Climbing	82	55%	260	15

Algorithm Comparison

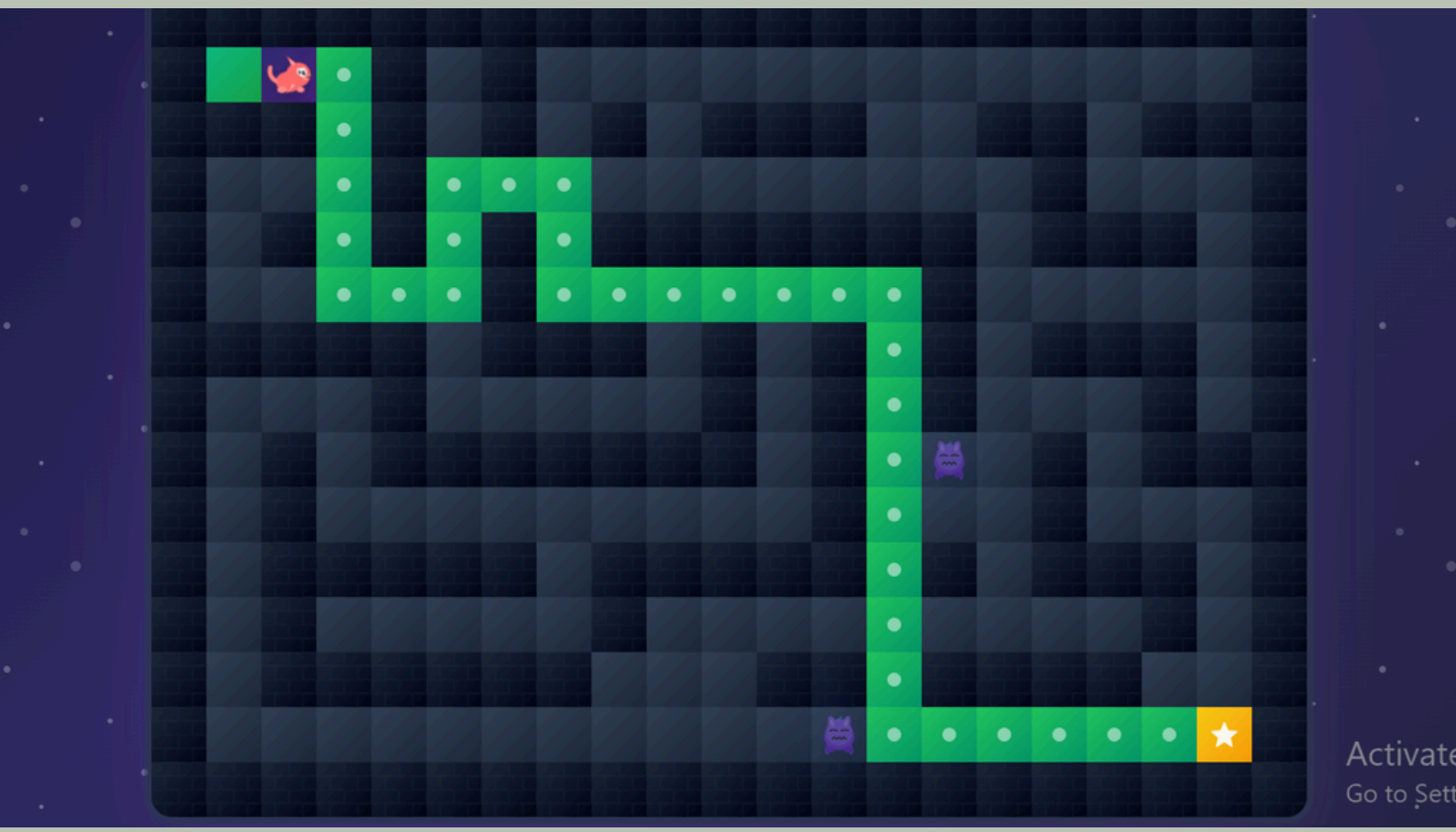
Scenario	Best Algorithm
Optimal & Guaranteed Path	A* (Manhattan)
Fastest Possible	Hill Climbing
Safe Navigation	A* (Danger-aware)
Simple Implementation	BFS
Not Recommended Overall	DFS

Recommendations

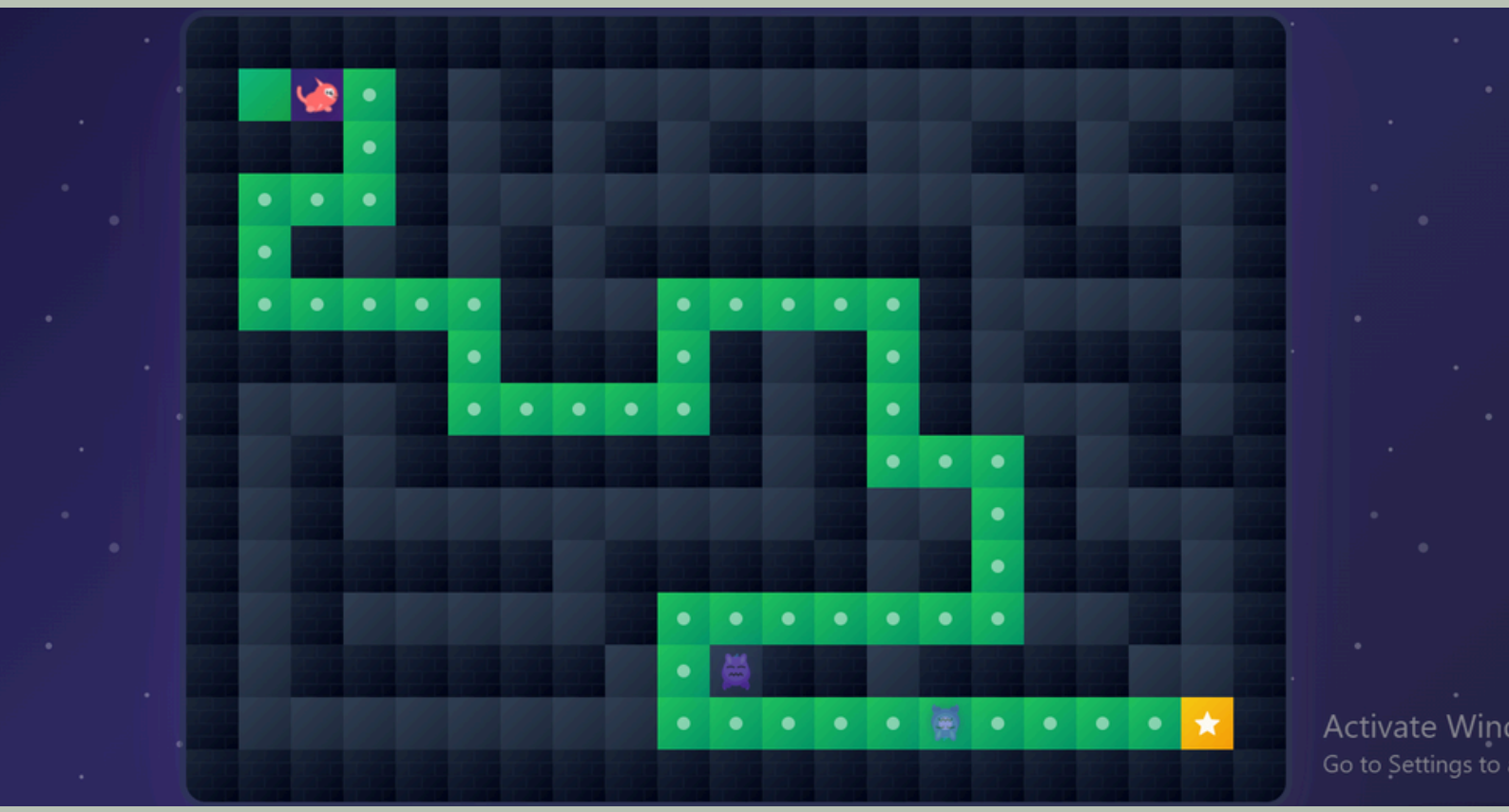
- **Replace repeated A* replanning with more efficient dynamic algorithms such as D Lite* or Lifelong Planning A*.**
- **Enhance monster behavior using predictive or cooperative movement strategies.**
- **Introduce Fog of War to simulate partially observable environments.**
- **Improve danger modeling by using dynamic or probabilistic risk costs instead of a fixed detection radius.**
- **Reduce replanning overhead through local planning and path reuse techniques.**
- **Upgrade visualization from a console-based view to a 2D graphical interface for clearer analysis.**

From here, we choose the algorithm that the team will use to reach the goal.

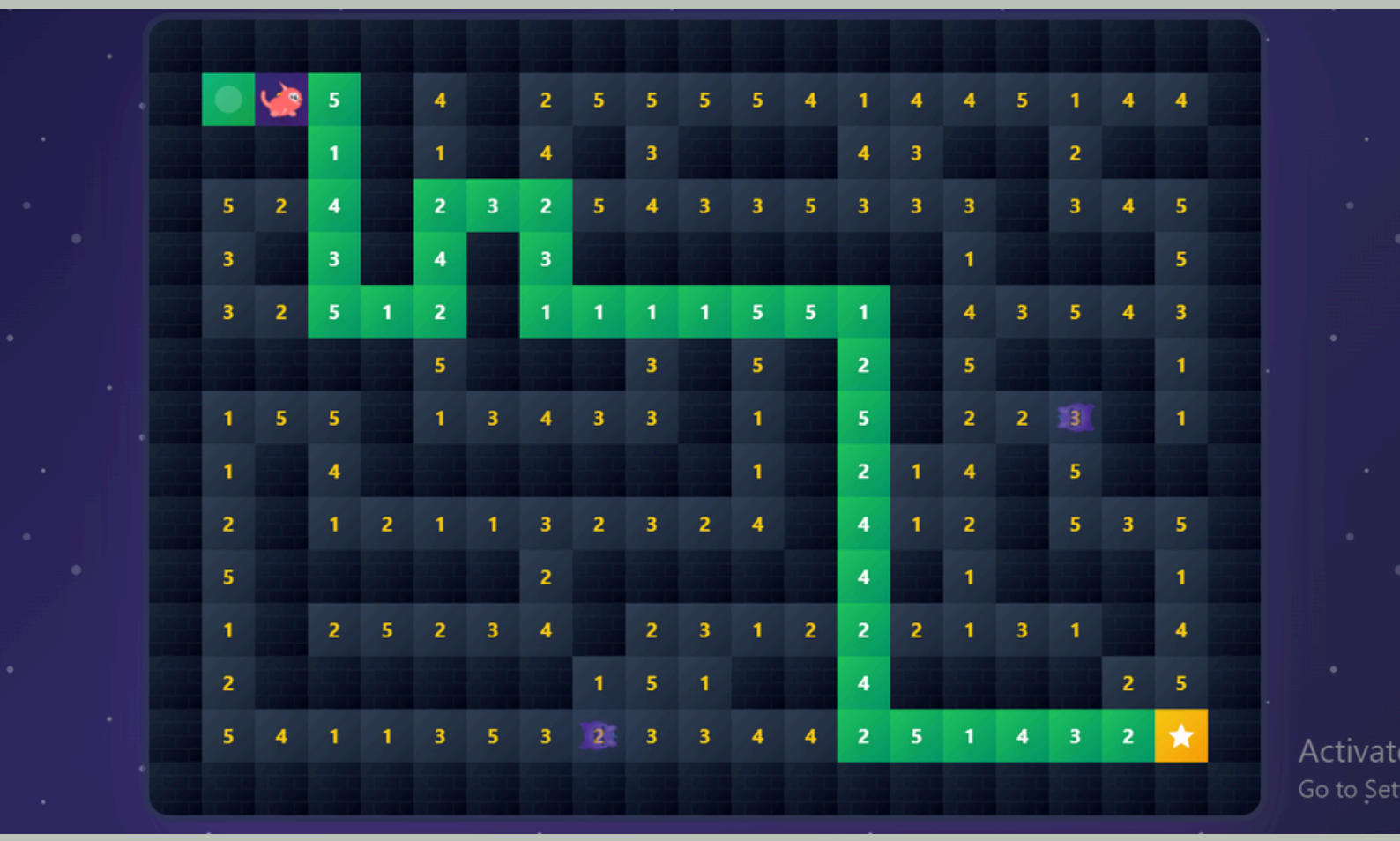
Breadth-First Search (BFS)



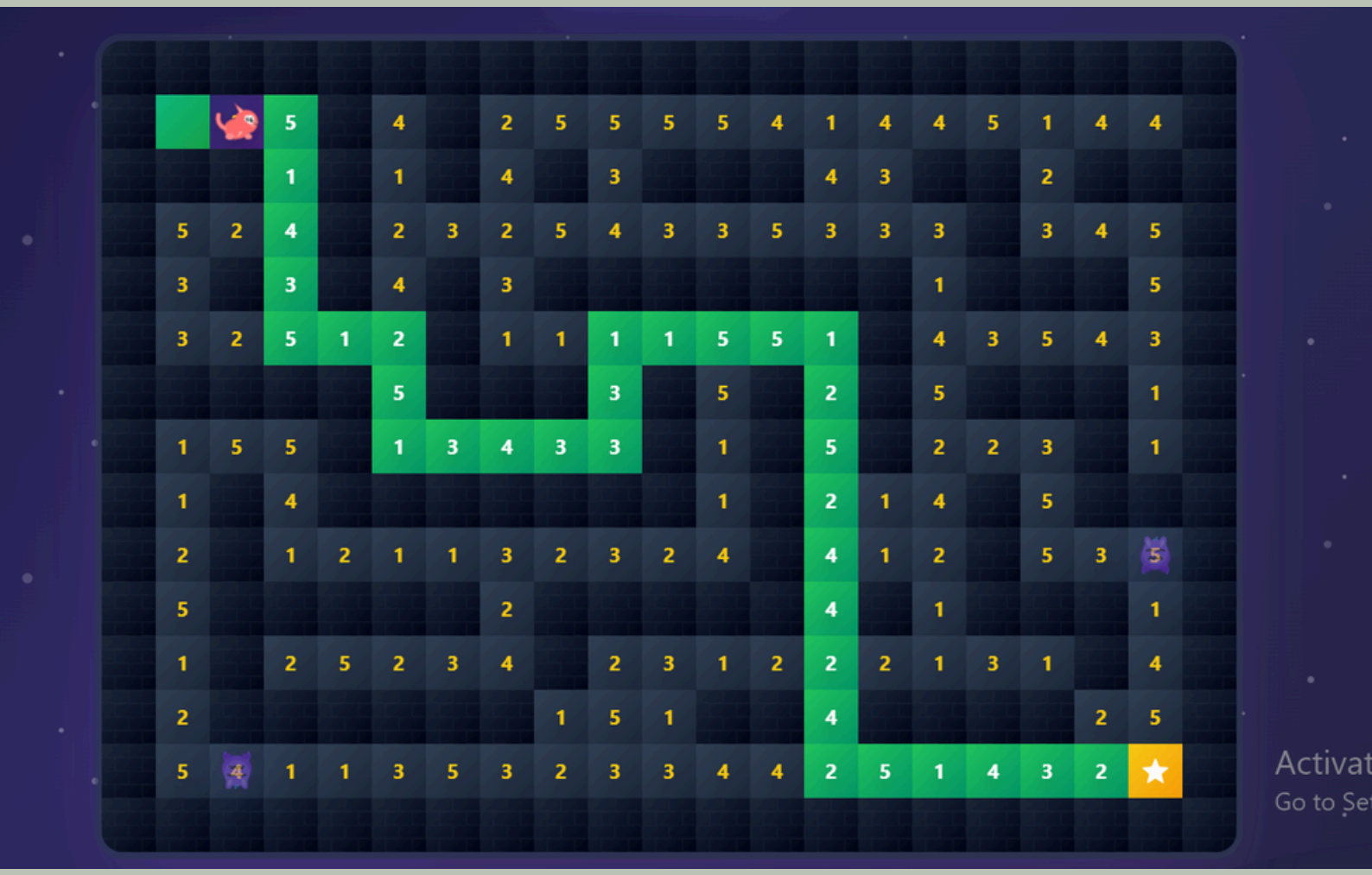
DFS (Depth-First Search)



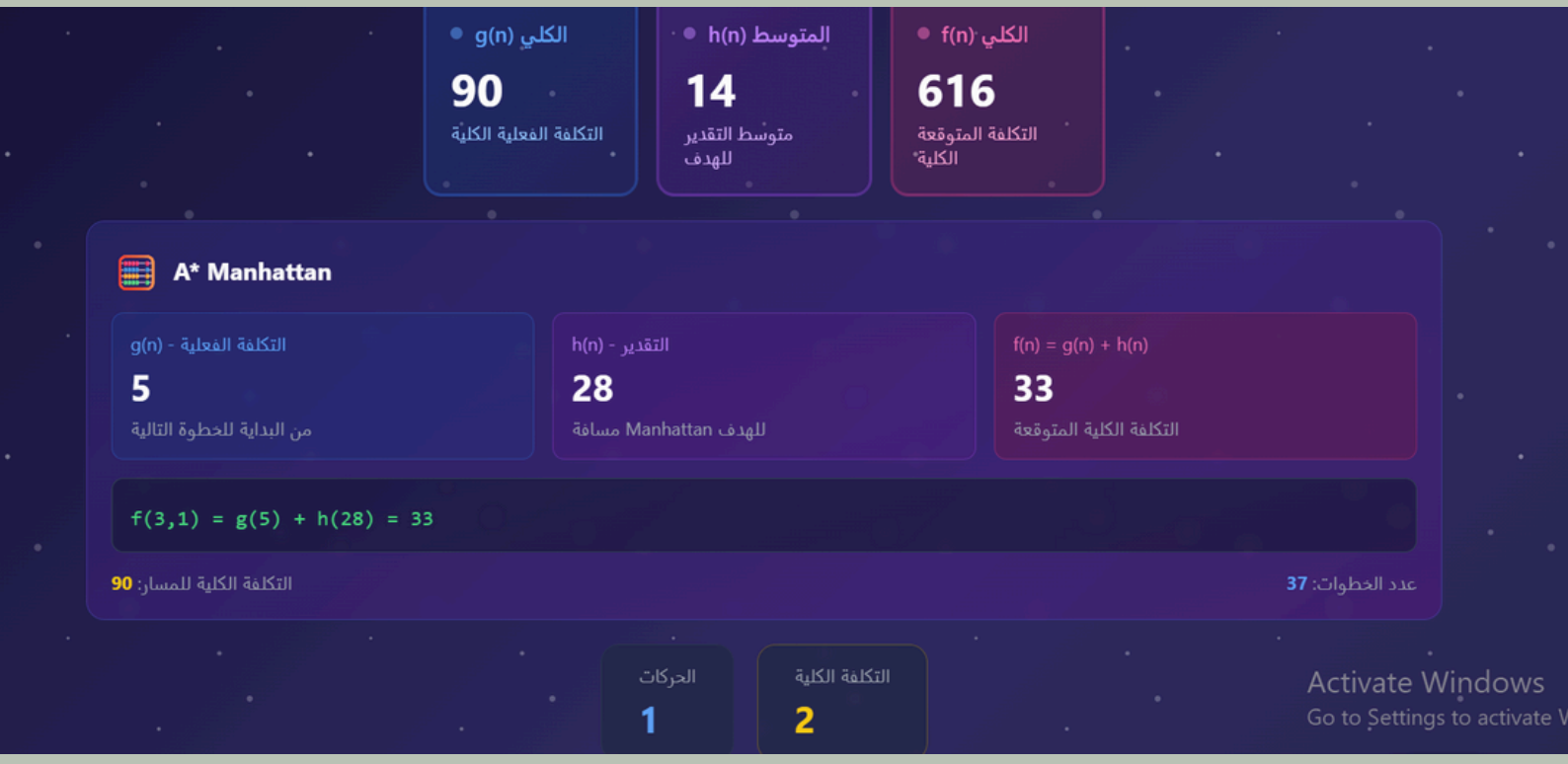
Uniform Cost Search (UCS)



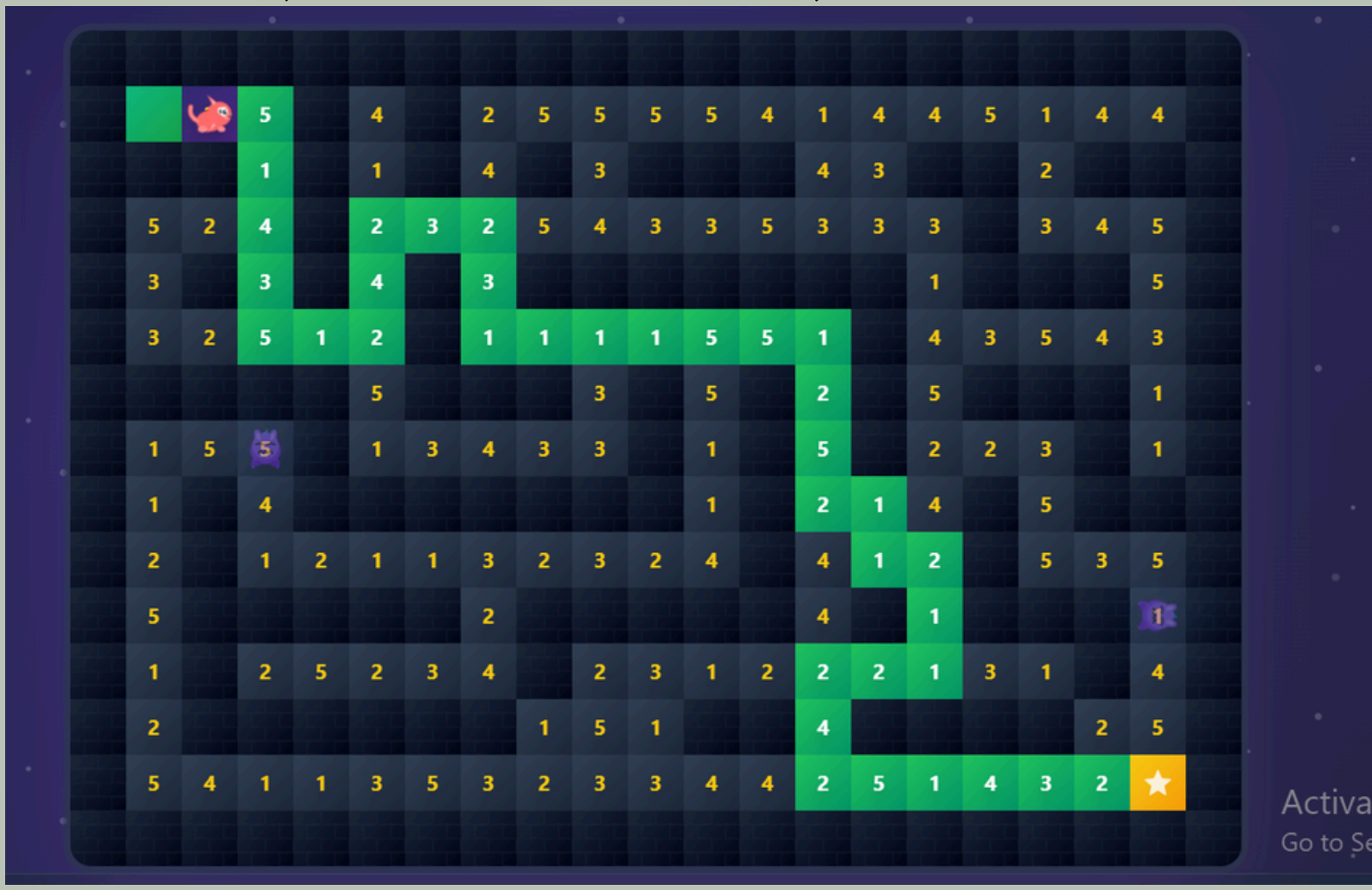
Iterative Deepening Search (IDS)



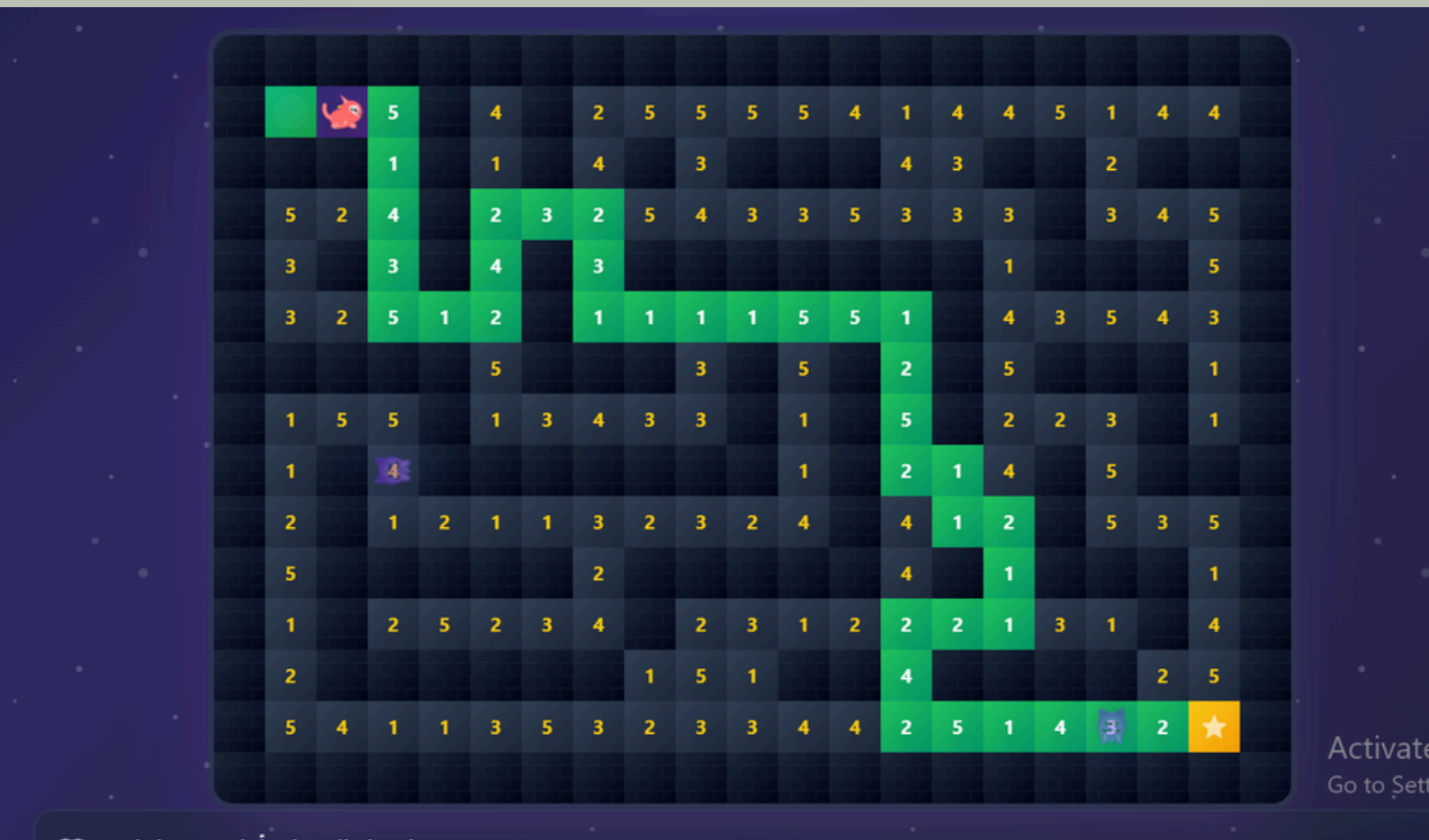
A* Search (Manhattan Distance) A* (Danger-Aware Heuristic)



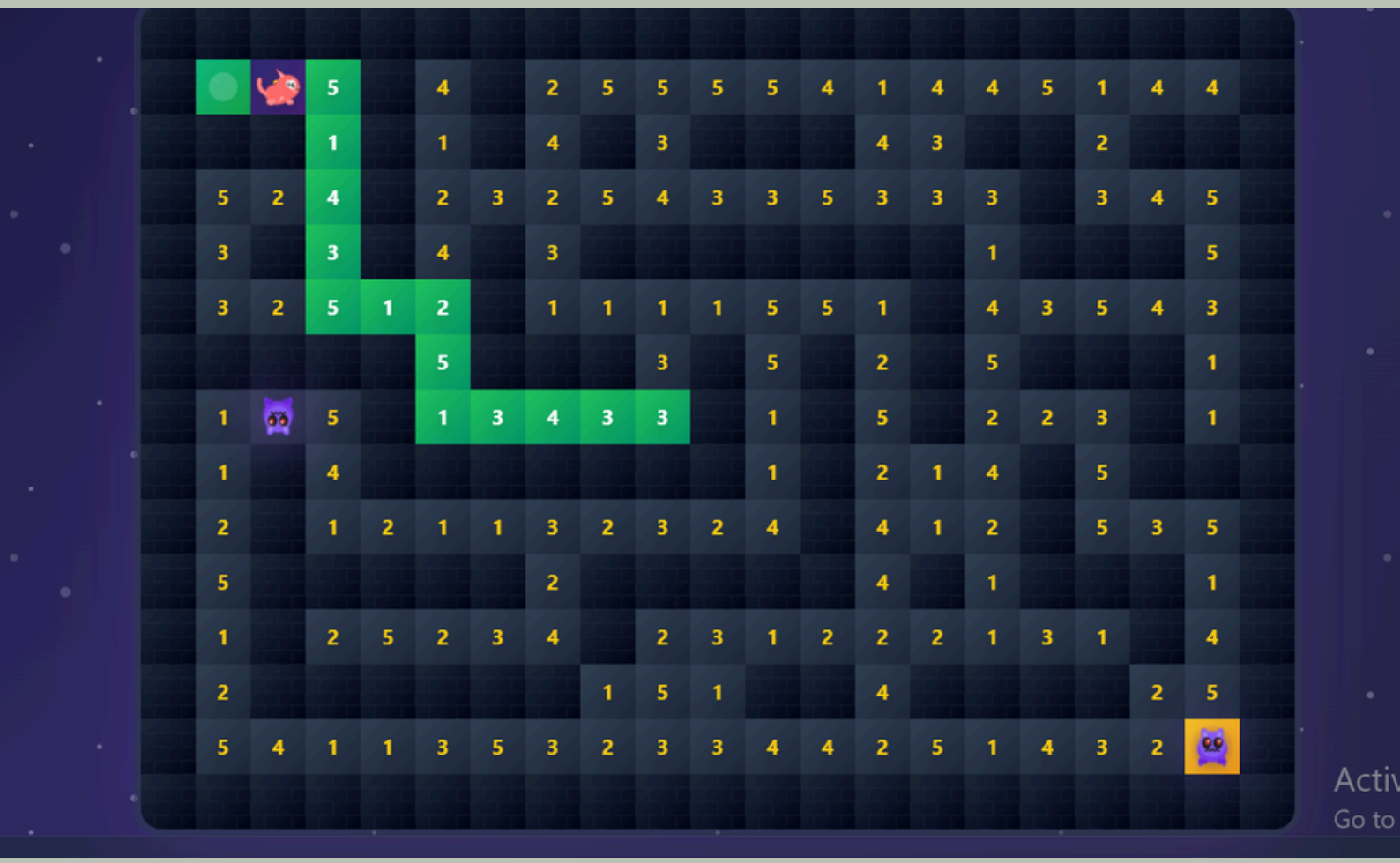
A* Search (Manhattan Distance)



A* (Danger-Aware Heuristic)



Hill Climbing Algorithm



visualization from console

```
🏰 Maze Generated                                   index-Blsj9HRn.js:64
📐 Size: 21x15                                      index-Blsj9HRn.js:64
🟢 Start: (1, 1)                                    index-Blsj9HRn.js:64
⭐ Exit: (19, 13)                                   index-Blsj9HRn.js:64
💰 Cost range: 1-5                                  index-Blsj9HRn.js:64
🔄 Multiple paths: Enabled (15% extra)              index-Blsj9HRn.js:64
                                                    index-Blsj9HRn.js:66
====================================
PLAYER: (1,1) -> (2,1) | Cost: 2                    index-Blsj9HRn.js:67
====================================                index-Blsj9HRn.js:67
Algorithm: BFS (Queue - FIFO)                       index-Blsj9HRn.js:67
Goal: Shortest path by STEPS                        index-Blsj9HRn.js:67
                                                    index-Blsj9HRn.js:67
Result:
  Steps: 33                                         index-Blsj9HRn.js:68
  Explored: 145 cells                               index-Blsj9HRn.js:68
  Path: R-D-D-D-D-R-...                             index-Blsj9HRn.js:68
                                                    index-Blsj9HRn.js:69
```

Map:

another step

Result:

Steps: 21

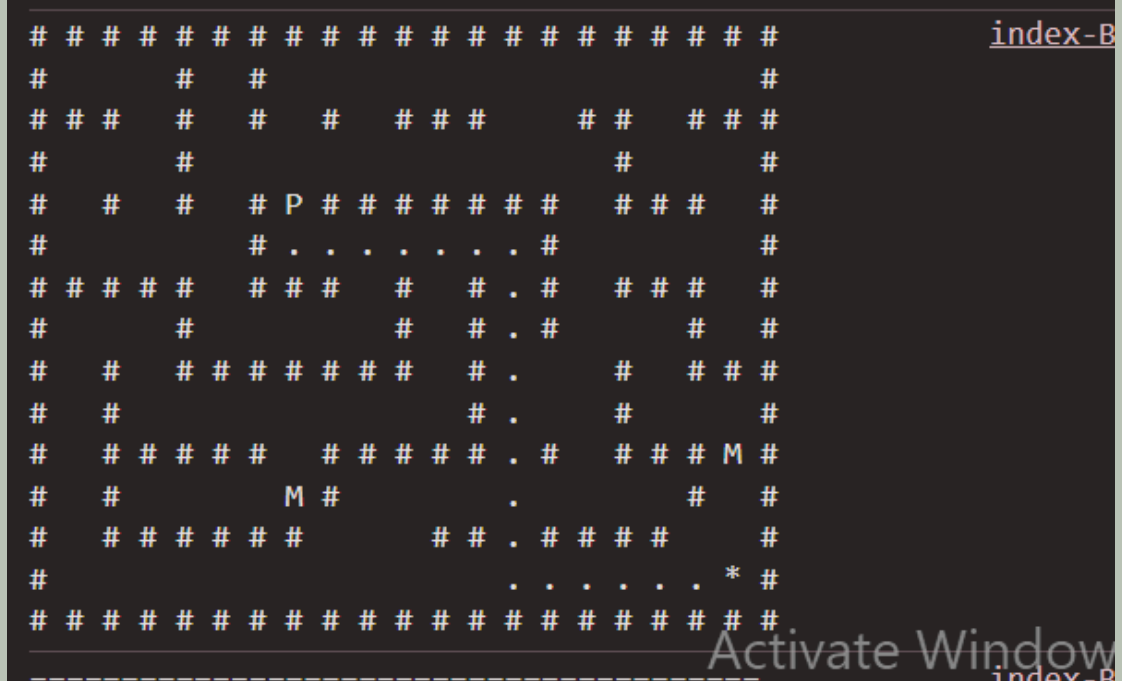
Explored: 114 cells

Total Cost: 55

Heuristic: Danger Aware (Monster Avoidance)

Path: D(h30)-R(h19)-R(h18)-R(h17)-R(h16)-R(h15)-...

Map:



Our Best Team

Mahmoud Mohamed

Ali Eid

MENNA TAMER

HABIBA KHATTAB

menna ahmed

Ahmed Abdelfatah

omar tallaat

omar ashraf