A* Pathfinding Algorithm Documentation & Pseudocode

Author: Ninad Metkar

Project Description:
This project implements the A* pathfinding algorithm to find the shortest path in a grid. It includes:
- Console output showing step-by-step exploration.
- Pygame visualization of visited nodes and final path.

Extra Features:
- Supports 8-direction movement (diagonal moves included).
- Human-like visualization with slightly random colors for visited nodes.
- Step-by-step messages in the console.

Assumptions:
- Grid is a 2D list: 0 = free cell, 1 = obstacle.
- Start and Goal positions are predefined.
- Movement allowed in 8 directions (can remove diagonals for 4-directional movement).
- If no path exists, an empty path is returned.
- Delays are added (time.sleep in console, clock.tick in Pygame) for smooth animation.
- Straight moves cost 1, diagonal moves cost 1.4.

Pseudocode:

```
Function Heuristic(node, goal):
    Return Euclidean distance between node and goal

Function A_Star(grid, start, goal):
    open_list = [(0, start)]
    came_from = {}
    g_score[start] = 0
    visited = []

    While open_list is not empty:
        current = node with lowest f_score
        Add current to visited
        Ifcurrent == goal:
            Reconstruct path from came_from
            Return visited, path

        Foreach neighbor of current (4 or 8 directions):
            Ifneighbor is valid and not obstacle:
                tentative_g = g_score[current] + movement_cost
                Iftentative_g < g_score[neighbor]:
                    Update g_score[neighbor]
                    came_from[neighbor] = current
                    Add neighbor to open_list with f = tentative_g + heuristic(neighbor, goal)

    Return visited, empty path
```

How It Works: 1. Start at the start node and explore neighbors using a priority queue (lowest f = g + h). 2. Console prints each visited node, Pygame shows it visually with light random shades. 3. Euclidean distance is used as the heuristic (works for diagonals). 4. When the goal is reached, the final path is reconstructed. 5. Symbols in console: S = Start, G = Goal, # = Obstacle, . = Visited, O = Path. 6. Pygame colors: green = start, red = goal, blue = path, light random = visited nodes.

How to Build & Run: 1. Install Pygame: pip install pygame
2. Save the script as astar_human.py.
3. Run the program: python astar_pygame_console.py
 4. Observe step-by-step exploration in console and smooth animation in Pygame.

Customization:
- Modify grid, start, or goal variables in the script.
- Adjust time.sleep in console or clock.tick in Pygame for animation speed.
- Remove diagonal moves in directions list for 4-directional movement.

Output Example:
Console:

Exploring node (0, 0) Exploring node (1, 0) ... Final Path: S . O . O # # . # . ... Legend: S=Start, G=Goal, #=Obstacle, .=Visited, O=Path

Pygame:
- Blue = Final path
- Light blue = Visited cells
- Green = Start
- Red = Goal
- Black = Obstacles

End of Documentation & Pseudocode