Problem statement : Pathfinding with A* algorithm

Name : Pragya
Branch : CSE-AIML
Roll No. : 202401100400135
Subject : Introduction to AI

Introduction

Pathfinding with A* algorithm
A (A-Star) Algorithm* is a well-known and effective pathfinding algorithm applied in AI, robotics, and game development. It calculates the shortest path from a starting position to an ending position with obstacles avoided.

How A Works*
A* employs a cost function to decide on the best path:
$f(n)=g(n)+h(n)f(n) = g(n) + h(n)f(n)=g(n)+h(n)$
Where:
$g(n)$ = Cost from the beginning node to node n (known cost).
$h(n)$ = Heuristic estimate of cost from n to goal (estimated cost).
$f(n)$ = Overall estimated cost (for node prioritization).
Important A* Features
✅ Will always find shortest path (if heuristic is admissible).
✅ Relatively quick compared to other uninformed search algorithms (such as Dijkstra's).
✅ Makes use of heuristic functions (Manhattan, Euclidean, or other distance metrics).
Uses of A*
Game AI (NPC movement, map pathfinding).
Robotics (autonomous navigation).
GPS Navigation (determining shortest paths).
Network Routing (finding best paths for data packets).

Methodology

Algorithm Steps
Set up an open list (priority queue) and a closed set.
Push the start node to the open list with $f = 0$.
Repeat until the open list is empty:
Get the node with the smallest f from the open list.
If it is the goal node, reconstruct the path and exit.
Otherwise, push it onto the closed set and consider its neighbors.
Calculate g, h, and f for every neighbor and insert valid ones into the open list.
If no path is discovered, return failure.

Heuristic Functions
Manhattan Distance ($|x1 - x2| + |y1 - y2|$) → Ideal for grid-based maps (no diagonal movement).
Euclidean Distance ($\mathrm{sqrt}((x1 - x2)^2 + (y1 - y2)^2)$) → Ideal for continuous space.
Diagonal Distance → Employed when there is diagonal movement.

Code

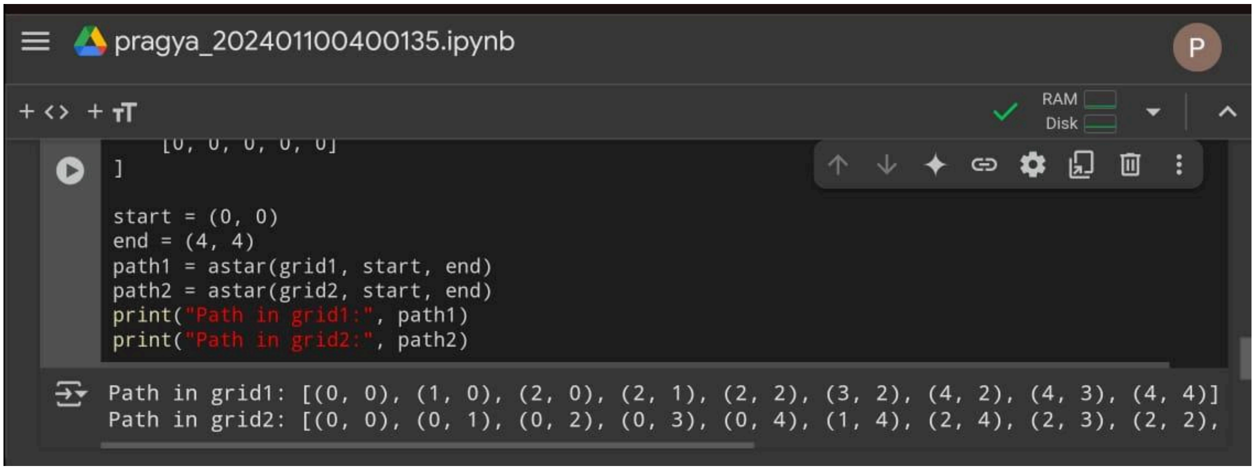
```
import heapq
class Node:
    def __init__(self, position, parent=None):
        self.position = position
        self.parent = parent
        self.g = 0
        self.h = 0
        self.f = 0
    def __lt__(self, other):
        return self.f < other.f

def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

def astar(grid, start, end):
    open_list = []
    closed_set = set()
    start_node = Node(start)
    end_node = Node(end)
    heapq.heappush(open_list, start_node)
    while open_list:
        current_node = heapq.heappop(open_list)
        if current_node.position == end:
            path = []
            while current_node:
                path.append(current_node.position)
                current_node = current_node.parent
            return path[::-1]
        closed_set.add(current_node.position)
        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            neighbor_pos = (current_node.position[0] + dx, current_node.position[1] + dy)
            if (neighbor_pos[0] < 0 or neighbor_pos[0] >= len(grid) or neighbor_pos[1] < 0 or neighbor_pos[1] >= len(grid[0]) or grid[neighbor_pos[0]][neighbor_pos[1]] == 1 or
neighbor_pos in closed_set):
                continue
            neighbor = Node(neighbor_pos, current_node)
            neighbor.g = current_node.g + 1
            neighbor.h = heuristic(neighbor.position, end)
            neighbor.f = neighbor.g + neighbor.h
            if any(n for n in open_list if n.position == neighbor.position and n.g <= neighbor.g):
                continue
            heapq.heappush(open_list, neighbor)
    return None

grid1 = [[0, 1, 0, 0, 0], [0, 1, 0, 1, 0], [0, 0, 0, 1, 0], [1, 1, 0, 1, 0], [0, 0, 0, 0, 0]]
grid2 = [[0, 0, 0, 0, 0], [1, 1, 1, 1, 0], [0, 0, 0, 0, 0], [0, 1, 1, 1, 1], [0, 0, 0, 0, 0]]
start = (0, 0)
end = (4, 4)
path1 = astar(grid1, start, end)
path2 = astar(grid2, start, end)
print("Path in grid1:", path1)
print("Path in grid2:", path2)
```

Output/Result



pragya_202401100400135.ipynb

```
    [0, 0, 0, 0, 0]
]

start = (0, 0)
end = (4, 4)
path1 = astar(grid1, start, end)
path2 = astar(grid2, start, end)
print("Path in grid1:", path1)
print("Path in grid2:", path2)
```

```
Path in grid1: [(0, 0), (1, 0), (2, 0), (2, 1), (2, 2), (3, 2), (4, 2), (4, 3), (4, 4)]
Path in grid2: [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 4), (2, 4), (2, 3), (2, 2),
```

References/Credits

1) Chat gpt: https://chatgpt.com
2) Wikipedia: Wikipedia