Pathfinding with A* Algorithm

Name	Anushika Sadhwani
Roll No	202401100400042
Date	10/03/2025

Introduction

Pathfinding is a fundamental problem in computer science and artificial intelligence, commonly used in robotics, gaming, and navigation systems. The A* (A-star) algorithm is one of the most efficient pathfinding algorithms, combining the benefits of Dijkstra’s Algorithm and Greedy Best-First Search. It finds the shortest path from a start node to a goal node using a heuristic function that balances the cost to reach a node and the estimated cost to the goal.

Scope and Methodology

The A* algorithm works by evaluating nodes based on the following function:

Steps of the algorithm:

1. Initialize an open list with the start node.
2. Maintain a closed list of visited nodes.
3. Extract the node with the lowest value.
4. Generate its neighbors and compute their costs.
5. If a neighbor is the goal, the path is found.
6. Otherwise, add valid neighbors to the open list and repeat.
7. Backtrack from the goal node to reconstruct the optimal path.

The heuristic function commonly used is the Manhattan distance (for grids) or Euclidean distance (for graphs).

Output/Code

```
import heapq

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

def astar(grid, start, goal):
    rows, cols = len(grid), len(grid[0])
    open_list = []
    heapq.heappush(open_list, (0, start))
    came_from = {}
    g_score = {node: float('inf') for row in grid for node in enumerate(row)}
    g_score[start] = 0
    f_score = {node: float('inf') for row in grid for node in enumerate(row)}
    f_score[start] = heuristic(start, goal)

    while open_list:
        _, current = heapq.heappop(open_list)
        if current == goal:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            return path[::-1]

        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            neighbor = (current[0] + dx, current[1] + dy)
            if 0 <= neighbor[0] < rows and 0 <= neighbor[1] < cols and
grid[neighbor[0]][neighbor[1]] == 0:
                tentative_g_score = g_score[current] + 1
                if tentative_g_score < g_score[neighbor]:
                    came_from[neighbor] = current
                    g_score[neighbor] = tentative_g_score
                    f_score[neighbor] = g_score[neighbor] + heuristic(neighbor,
goal)
                    heapq.heappush(open_list, (f_score[neighbor], neighbor))
    return []

# Example grid (0 = free space, 1 = obstacle)
grid = [[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]]

start, goal = (0, 0), (4, 4)
path = astar(grid, start, goal)
print("Path found:", path)
```

Output/Result

(Sample Output)

```
Path found: [(0, 0), (1, 0), (2, 0), (2, 1), (2, 2), (3, 2), (4,
```

(Insert screenshots of the program running with a graphical representation if available)

References/ Credits

- A* Algorithm documentation and research papers.
- Python documentation for the heapq module.
- Online pathfinding resources and tutorials.
- (Include any additional sources used)