KIET
GROUP OF INSTITUTIONS
Connecting Life with Learning

Project Title: Pathfinding with A* Algorithm

Student Name: Akul Sharma

Roll No: 202401100300027

Faculty Supervisor: Bikki Gupta

Institution: KIET, CSE AI

2. Introduction Pathfinding algorithms are essential in artificial intelligence, robotics, and game development. The A* (A-Star) algorithm is one of the most efficient pathfinding techniques used to find the shortest route between a start and a goal node in a grid-based environment. This report provides an overview of the A* algorithm, its implementation in Python, and an example application.

3. Methodology A* is a best-first search algorithm that uses a combination of:

- **G-cost**: The cost to move from the start node to the current node.
- **H-cost**: The heuristic estimate of the cost from the current node to the goal.
- **F-cost**: The sum of G-cost and H-cost, i.e., F = G + H.

The algorithm prioritizes nodes with the lowest F-cost, ensuring an optimal and efficient path to the goal.

The A* algorithm is implemented in Python using a priority queue (heap) to manage open nodes efficiently. The heuristic function used is the Manhattan distance, suitable for grid-based movement.

4. Code Implementation

```
import heapq


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


def a_star(grid, start, goal):
    rows, cols = len(grid), len(grid[0])
    open_set = []
    heapq.heappush(open_set, (0, start))
    came_from = {}
    g_score = {start: 0}
    f_score = {start: heuristic(start, goal)}


    while open_set:
        _, current = heapq.heappop(open_set)
```

```
        if current == goal:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            path.reverse()
            return path

        neighbors = [(0, 1), (1, 0), (0, -1), (-1, 0)]
        for dx, dy in neighbors:
            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 neighbor not in g_score or tentative_g_score < g_score[neighbor]:
                    came_from[neighbor] = current
                    g_score[neighbor] = tentative_g_score
                    f_score[neighbor] = tentative_g_score + heuristic(neighbor, goal)
                    heapq.heappush(open_set, (f_score[neighbor], neighbor))

    return None  # No path found
```

5. Output/Result The algorithm is tested on a 5x5 grid with obstacles. The function returns the shortest path from the start (0,0) to the goal (4,4), avoiding obstacles.

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

Screenshot of Output:
[Insert Screenshot Here]

6. References/Credits

- A* Algorithm Documentation: https://en.wikipedia.org/wiki/A*_search_algorithm
- Python Heapq Library: https://docs.python.org/3/library/heapq.html
- Additional resources and datasets used in the project.

7. Conclusion The A* algorithm efficiently finds the shortest path in a grid by balancing cost and heuristic estimation. It is widely used in various applications such as robotics navigation, AI pathfinding, and game development. The provided implementation demonstrates its effectiveness in a simple grid-based environment.

8. Github link: https://github.com/akull07/akulsharma_.git