Pathfinding with A* Algorithm

Name: Mahadevan Pillai

Class : CSEAIML (B)

Roll No: 202401100400115

Course: AI

Introduction:

Pathfinding is a crucial aspect of artificial intelligence and robotics, widely used in gaming, GPS navigation, and robotics. 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 starting point to a target while considering both the cost to reach a node and the estimated cost to the goal.

A* operates on a graph representation where each node has a cost function f(n) = g(n) + h(n), where:

- g(n): The actual cost from the start node to the current node.
- h(n): The heuristic estimated cost from the current node to the goal.

A* uses a priority queue to explore the most promising path first, making it an optimal and complete algorithm for many applications.

Methodology:

1. Define the Grid or Graph: Represent the environment as a grid or a weighted graph.
2. Initialize Open and Closed Lists:
    - Open list: Nodes to be evaluated.
    - Closed list: Nodes already evaluated.
3. Start from the Initial Node: Calculate its f(n) = g(n) + h(n) and push it into the open list.
4. Select the Best Node: Choose the node with the lowest f(n) value from the open list.
5. Expand Neighbors: Compute their cost values and update paths accordingly.
6. Repeat Until Goal is Reached: Continue the process until the goal node is selected from the open list.
7. Backtrack to Find the Path: Trace back from the goal node to the start node to obtain the optimal path.

CODE:

```
import heapq

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

def a_star_search(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)}

    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]

    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)
            return path[::-1]  # Return reversed path
```

```
    for dx, dy in directions:
        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

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

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

```python
grid = [
    [0, 1, 0, 0, 0],
    [0, 1, 0, 1, 0],
    [0, 0, 0, 1, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 0, 0, 0]
]

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

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

References/Credits:

- Wikipedia: A* search algorithm
- Online tutorials and documentation on pathfinding algorithms.
- ChatGPT: Code