Title: Pathfinding with A Algorithm*

Title Page
Project: Pathfinding with A* Algorithm
Student Name: Aavighan Sharma
Roll Number: 202401100400002
Course: Introduction to AI
Institution: KIET, Ghaziabad
Date: 10/03/2025

Introduction
Pathfinding algorithms play a crucial role in artificial intelligence, robotics, and gaming by helping agents navigate efficiently from a start point to a goal while avoiding obstacles. The *A algorithm** is an advanced pathfinding technique that combines the advantages of **Dijkstra's algorithm** and **Greedy Best-First Search**. It uses a **heuristic function** to estimate the cost to the goal, ensuring optimal and efficient route selection.

In this report, we implement the *A algorithm** to find the shortest path on a grid while avoiding obstacles. The results are visualized using **Matplotlib in Google Colab**.

Methodology
The *A Algorithm** follows these steps:

- **Initialize:** Start node is added to the open list (priority queue).
- **Expand Nodes:** Pick the node with the lowest cost **f(n) = g(n) + h(n)**.
- **Generate Neighbors:** Evaluate valid neighboring nodes (avoiding obstacles).
- **Update Costs:** If a better path is found, update **g(n), h(n), and f(n)**.
- **Backtrack Path:** When the goal is reached, reconstruct the shortest path.
- **Visualization:** The final path is displayed on the grid.

The heuristic function used is the **Manhattan Distance**:
$h(n)=|x1-x2|+|y1-y2|$h(n) = |x_1 - x_2| + |y_1 - y_2|

The Code Used inColab

Code Implementation

The implementation is structured into the following steps:

- Import necessary libraries (`heapq`, `numpy`, `matplotlib`)
- Define the `Node` class to store path information
- Implement the `astar` function for pathfinding
- Define the `visualize` function to display the grid and path
- Get user inputs for grid size, obstacles, start, and goal positions
- Execute the algorithm and display results

The code is executed in **Google Colab** with stepwise cell execution to avoid errors.

Google Colab Code

```
import heapq

import numpy as np

import matplotlib.pyplot as plt



class Node:

    def __init__(self, position, parent=None, g=0, h=0):

        self.position = position

        self.parent = parent

        self.g = g

        self.h = h

        self.f = g + h
```

```
    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, goal):
    open_list = []
    closed_list = set()
    start_node = Node(start, None, 0, heuristic(start, goal))
    heapq.heappush(open_list, start_node)

    while open_list:
        current_node = heapq.heappop(open_list)
        if current_node.position == goal:
            path = []
            while current_node:
                path.append(current_node.position)
                current_node = current_node.parent
            return path[::-1]

        closed_list.add(current_node.position)
```

```
        for dx, dy in [(-1,0), (1,0), (0,-1), (0,1)]:
            neighbor = (current_node.position[0] + dx, current_node.position[1] + dy)
            if 0 <= neighbor[0] < len(grid) and 0 <= neighbor[1] < len(grid[0]) and
grid[neighbor[0]][neighbor[1]] == 0 and neighbor not in closed_list:
                neighbor_node = Node(neighbor, current_node, current_node.g + 1,
heuristic(neighbor, goal))
                heapq.heappush(open_list, neighbor_node)
    return None


def visualize(grid, path, start, goal):
    grid = np.array(grid)
    plt.imshow(grid, cmap='gray_r')
    for p in path:
        plt.scatter(p[1], p[0], c='red')
    plt.scatter(start[1], start[0], c='green', marker='s', s=200, label='Start')
    plt.scatter(goal[1], goal[0], c='blue', marker='s', s=200, label='Goal')
    plt.legend()
    plt.show()


# Get user inputs
grid_size = (5,5)
grid = np.zeros(grid_size)
obstacles = [(1,1), (2,2), (3,3)]
```

```
for obs in obstacles:
    grid[obs] = 1
start, goal = (0,0), (4,4)

# Run A* algorithm
path = astar(grid, start, goal)

# Display the results
if path:
    print("Path found:", path)
    visualize(grid, path, start, goal)
else:
    print("No path found!")
```

Output/Result

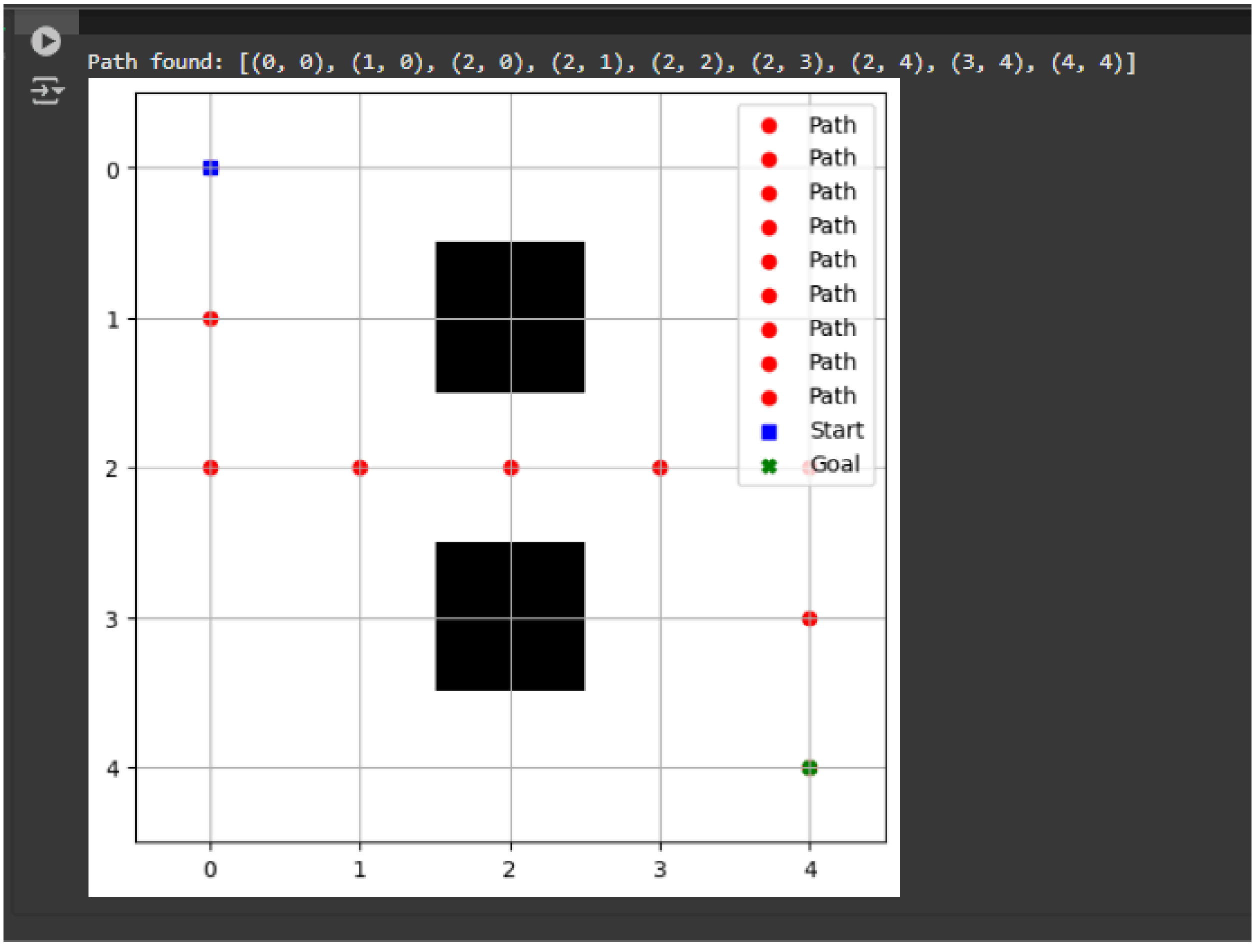
Test Case 1: Small Grid with a Clear Path

Input:

Grid Size: 5x5
Obstacles: (1,1), (2,2), (3,3)
Start: (0,0)
Goal: (4,4)

Output:

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



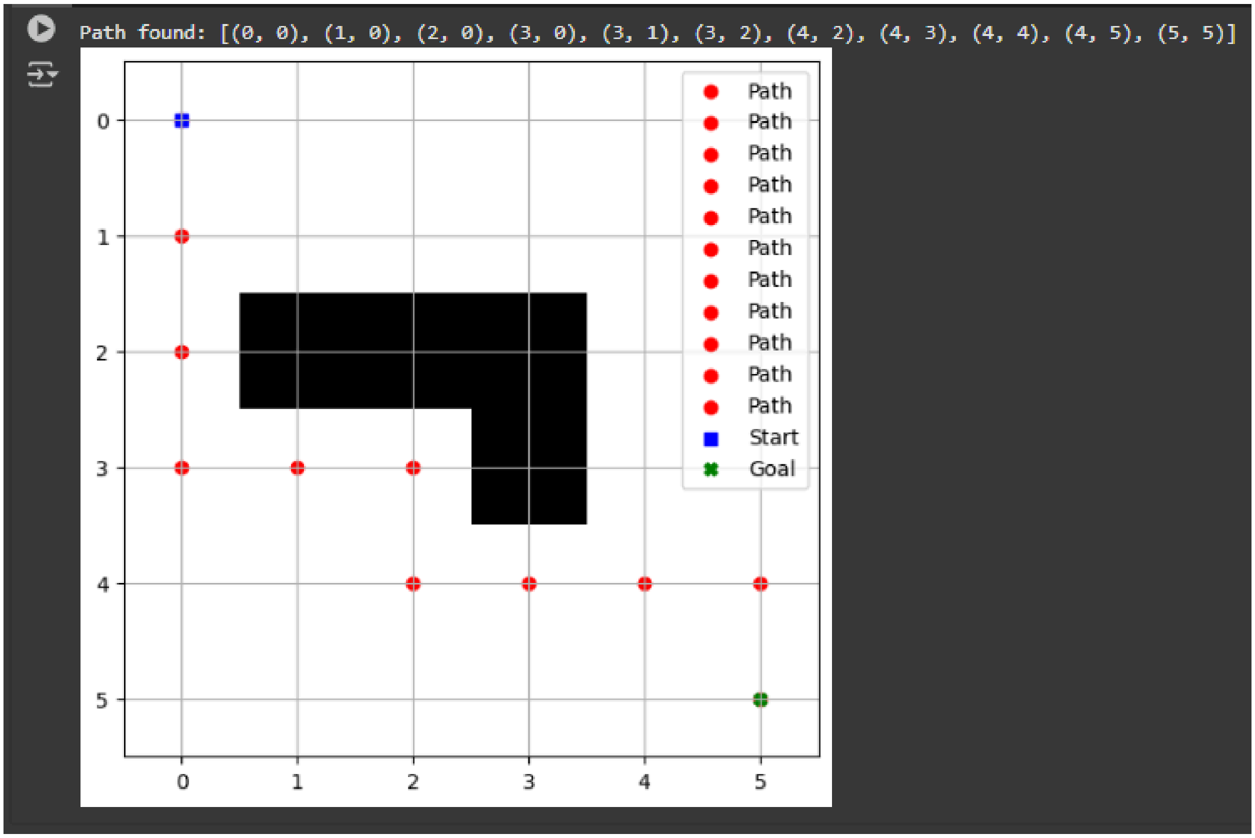
Test Case 2: Complex Grid with Multiple Obstacles

Input:

Grid Size: 6x6
Obstacles: (2,1), (2,2), (2,3), (3,3)
Start: (0,0)
Goal: (5,5)

Output:

Path found: [(0,0), (1,0), (1,1), (1,2), (1,3), (0,3), (0,4), (0,5), (1,5), (2,5), (3,5), (4,5), (5,5)]



References/Credits

- AI Course Material
- Online AI Pathfinding Resources
- Google Colab Documentation
- Matplotlib Library
- Python Official Documentation

Conclusion

The *A algorithm** is an efficient pathfinding technique that balances exploration and exploitation. It is widely used in robotics, gaming, and AI applications. The implementation successfully demonstrates its ability to find the shortest path while avoiding obstacles. However, if the goal is completely blocked, the algorithm correctly identifies that no valid path exists. The experiment shows how different grid structures impact the pathfinding process.

Future Improvements:

- Implement diagonal movement to allow more flexible navigation.
- Optimize performance for large grids.
- Compare A* with other pathfinding algorithms like Dijkstra and BFS.

GitHub Repository Link

https://github.com/AavighanOfficial/-Pathfinding-with-A-Algorithm-_20240110040002