**Practicle 2**

Implement A star (A\*) Algorithm for any game search problem

**Code:**

import heapq

class AStar:

def \_\_init\_\_(self, grid, start, goal):

self.grid = grid # 2D grid where 0 = walkable, 1 = blocked

self.start = start # Start position (x, y)

self.goal = goal # Goal position (x, y)

self.rows = len(grid)

self.cols = len(grid[0])

def heuristic(self, node):

# Manhattan distance heuristic

return abs(node[0] - self.goal[0]) + abs(node[1] - self.goal[1])

def neighbors(self, node):

# Return valid neighbors (up, down, left, right)

dirs = [(0, 1), (1, 0), (0, -1), (-1, 0)] # Directions: right, down, left, up

result = []

for d in dirs:

neighbor = (node[0] + d[0], node[1] + d[1])

if 0 <= neighbor[0] < self.rows and 0 <= neighbor[1] < self.cols and self.grid[neighbor[0]][neighbor[1]] == 0:

result.append(neighbor)

return result

def a\_star\_search(self):

# Priority queue to store (f\_score, node)

open\_list = []

heapq.heappush(open\_list, (0, self.start))

came\_from = {} # For reconstructing path

g\_score = {self.start: 0} # Cost from start to each node

f\_score = {self.start: self.heuristic(self.start)} # Estimated cost from start to goal

while open\_list:

current = heapq.heappop(open\_list)[1]

# If we reached the goal, reconstruct the path

if current == self.goal:

return self.reconstruct\_path(came\_from, current)

for neighbor in self.neighbors(current):

tentative\_g\_score = g\_score[current] + 1 # Distance from current to neighbor is 1

if neighbor not in g\_score or tentative\_g\_score < g\_score[neighbor]:

# Update the best path to the neighbor

came\_from[neighbor] = current

g\_score[neighbor] = tentative\_g\_score

f\_score[neighbor] = tentative\_g\_score + self.heuristic(neighbor)

heapq.heappush(open\_list, (f\_score[neighbor], neighbor))

return [] # Return empty path if no solution

def reconstruct\_path(self, came\_from, current):

# Reconstruct path from came\_from dictionary

total\_path = [current]

while current in came\_from:

current = came\_from[current]

total\_path.append(current)

return total\_path[::-1] # Reverse path to start from the beginning

# 0 = walkable, 1 = blocked

grid = [

[0, 1, 0, 0, 0],

[0, 1, 0, 1, 0],

[0, 0, 0, 1, 0],

[1, 1, 0, 0, 0],

[0, 0, 0, 1, 0]

]

start = (0, 0) # Starting position

goal = (4, 4) # Goal position

a\_star = AStar(grid, start, goal)

path = a\_star.a\_star\_search()

print("Path from start to goal:", path)