# Assessment 2: A\* Search Algorithm

**Step-by-Step Instructions with Comments in Python Code**

import heapq

import math

# Define the A\* search algorithm

def a\_star\_search(maze, start, goal, heuristic):

rows, cols = len(maze), len(maze[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:

return reconstruct\_path(came\_from, current)

for neighbor in get\_neighbors(current, maze, rows, cols):

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)

if neighbor not in [i[1] for i in open\_set]:

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

return []

# Helper to get valid neighbors

def get\_neighbors(node, maze, rows, cols):

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

neighbors = []

for dr, dc in directions:

r, c = node[0] + dr, node[1] + dc

if 0 <= r < rows and 0 <= c < cols and maze[r][c] == 0:

neighbors.append((r, c))

return neighbors

# Heuristic functions

def manhattan\_distance(node, goal):

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

def euclidean\_distance(node, goal):

return math.sqrt((node[0] - goal[0])\*\*2 + (node[1] - goal[1])\*\*2)

# Helper to reconstruct path

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

path = [current]

while current in came\_from:

current = came\_from[current]

path.append(current)

return path[::-1]

# Test the implementation

maze = [

[0, 0, 1, 0],

[0, 1, 0, 0],

[0, 0, 0, 1],

[1, 0, 0, 0]

]

start = (0, 0)

goal = (3, 3)

# Using Manhattan Distance

path\_manhattan = a\_star\_search(maze, start, goal, manhattan\_distance)

print("Path with Manhattan Distance:", path\_manhattan)

# Using Euclidean Distance

path\_euclidean = a\_star\_search(maze, start, goal, euclidean\_distance)

print("Path with Euclidean Distance:", path\_euclidean)