Practical 4: Astar

Q1) Demonstrate the Astar Algorithm.

Ans:

p4\_astar.py

"""

p4\_astar.py

Author: Jagrut Gala

Date: 07-08-2021

Practical: 4

Objective: Demonstrate Astar Algorithm

"""

class Node():

"""A node class for A\* Pathfinding"""

def \_\_init\_\_(self, parent=None, position=None):

self.parent = parent

self.position = position

self.g = 0

self.h = 0

self.f = 0

def \_\_eq\_\_(self, other):

return self.position == other.position

def astar(maze, start, end):

"""Returns a list of tuples as a path from the given start to the given end in the given maze"""

# Create start and end node

start\_node = Node(None, start)

start\_node.g = start\_node.h = start\_node.f = 0

end\_node = Node(None, end)

end\_node.g = end\_node.h = end\_node.f = 0

# Initialize both open and closed list

open\_list = []

closed\_list = []

# Add the start node

open\_list.append(start\_node)

# Loop until you find the end

while len(open\_list) > 0:

# Get the current node

current\_node = open\_list[0]

current\_index = 0

for index, item in enumerate(open\_list):

if item.f < current\_node.f:

current\_node = item

current\_index = index

# Pop current off open list, add to closed list

open\_list.pop(current\_index)

closed\_list.append(current\_node)

# Found the goal

if current\_node == end\_node:

path = []

current = current\_node

while current is not None:

path.append(current.position)

current = current.parent

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

# Generate children

children = []

for new\_position in [(0, -1), (0, 1), (-1, 0), (1, 0), (-1, -1), (-1, 1), (1, -1), (1, 1)]: # Adjacent squares

# Get node position

node\_position = (current\_node.position[0] + new\_position[0], current\_node.position[1] + new\_position[1])

# Make sure within range

if node\_position[0] > (len(maze) - 1)\

or node\_position[0] < 0\

or node\_position[1] > (len(maze[len(maze)-1]) -1)\

or node\_position[1] < 0:

continue

# Make sure walkable terrain

if maze[node\_position[0]][node\_position[1]] != 0:

continue

# Create new node

new\_node = Node(current\_node, node\_position)

# Append

children.append(new\_node)

# Loop through children

for child in children:

# Child is on the closed list

for closed\_child in closed\_list:

if child == closed\_child:

continue

# Create the f, g, and h values

child.g = current\_node.g + 1

child.h = ((child.position[0] - end\_node.position[0]) \*\* 2) + ((child.position[1] - end\_node.position[1]) \*\* 2)

child.f = child.g + child.h

# Child is already in the open list

for open\_node in open\_list:

if child == open\_node and child.g > open\_node.g:

continue

# Add the child to the open list

open\_list.append(child)

def main():

maze = [[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

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

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

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

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

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

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

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

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

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

start = (0, 0)

end = (7, 6)

path = astar(maze, start, end)

print(path)

if \_\_name\_\_ == '\_\_main\_\_':

main()

