**LAB 2 – ASSIGNMENT**

**SOLVE THE RAT IN A MAZE PROBLEM WITH VISUALIZATION**

**QUESTION-2. MAZE = [**

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

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

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

**[1, 1, 1, 1]**

**]**

CODE FOR DFS:

#*USING DFS QUESTION 2*

class MazeSolver:

def \_\_init\_\_(self, maze):

self.maze = maze

self.rows = len(maze)

self.cols = len(maze[0])

self.visited = [[False] \* self.cols *for* \_ *in* range(self.rows)]

self.solution = [[0] \* self.cols *for* \_ *in* range(self.rows)]

def *is\_valid\_move*(self, row, col):

*return* 0 <= row < self.rows and 0 <= col < self.cols and not self.visited[row][col] and self.maze[row][col] == 1

def *depth\_first\_search*(self, row, col):

*if* row == self.rows - 1 and col == self.cols - 1:

self.solution[row][col] = 1 # *Mark the destination cell*

*return* True

*if* self.is\_valid\_move(row, col):

self.visited[row][col] = True

self.solution[row][col] = 1

# *Explore in all four directions: up, down, left, right*

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

*for* dr, dc *in* directions:

*if* self.depth\_first\_search(row + dr, col + dc):

*return* True

# *If no valid move found, backtrack*

self.solution[row][col] = 0

*return* False

def *solve\_maze*(self):

*if* not self.depth\_first\_search(0, 0):

print("No solution exists.")

*else*:

self.print\_solution()

def *print\_solution*(self):

*for* row *in* self.solution:

print(" ".join(map(str, row)))

# *question 2(written from the board)*

maze = [

[1, 0, 0, 0],

[1, 1, 0, 1],

[0, 1, 0, 0],

[1, 1, 1, 1]

]

# *We use this method to solve the given question 2*

solver = MazeSolver(maze)

solver.solve\_maze()

# *we make a dfs function to solve this with dfs*

def *dfs*(maze, start, goal):

rows, cols = len(maze), len(maze[0])

directions = [(-1, 0), (1, 0), (0, -1), (0, 1)] # *Up, Down, Left, Right*

stack = [start]

visited = set()

visited.add(start)

parent = {start: None}

*while* stack:

current = stack.pop()

*if* current == goal:

*break*

*for* dr, dc *in* directions:

nr, nc = current[0] + dr, current[1] + dc

*if* 0 <= nr < rows and 0 <= nc < cols and maze[nr][nc] == 1 and (nr, nc) not in visited: # *'1' indicates a valid path*

stack.append((nr, nc))

visited.add((nr, nc))

parent[(nr, nc)] = current

# *Reconstruct the path*

path = []

step = goal

*while* step:

path.append(step)

step = parent.get(step)

path.reverse()

*return* path

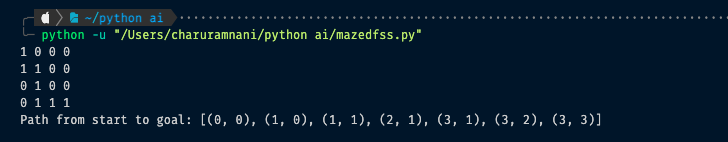
start = (0, 0) # *Starting point*

goal = (3, 3) # *Goal point*

path = dfs(maze, start, goal)

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

OUTPUT:



CODE FOR DFS VISUALIZATION:

import matplotlib.pyplot as plt

import matplotlib.animation as animation

class MazeSolver:

def \_\_init\_\_(self, maze):

self.maze = maze

self.rows = len(maze)

self.cols = len(maze[0])

self.visited = [[False] \* self.cols for \_ in range(self.rows)]

self.solution = [[0] \* self.cols for \_ in range(self.rows)]

self.path = [] # To store the path of the rat

def is\_valid\_move(self, row, col):

return 0 <= row < self.rows and 0 <= col < self.cols and not self.visited[row][col] and self.maze[row][col] == 1

def depth\_first\_search(self, row, col):

# If the goal is reached, stop searching

if row == self.rows - 1 and col == self.cols - 1:

self.solution[row][col] = 1

self.path.append((row, col))

return True

if self.is\_valid\_move(row, col):

self.visited[row][col] = True

self.solution[row][col] = 1

self.path.append((row, col))

# Explore in all four directions: up, down, left, right

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

for dr, dc in directions:

if self.depth\_first\_search(row + dr, col + dc):

return True

# If no valid move found, backtrack

self.solution[row][col] = 0

self.path.pop()

return False

def solve\_maze(self):

if not self.depth\_first\_search(0, 0):

print("No solution exists.")

else:

self.print\_solution()

def print\_solution(self):

for row in self.solution:

print(" ".join(map(str, row)))

def animate\_solution(solver):

fig, ax = plt.subplots()

ax.imshow(solver.maze, cmap="Greys", vmin=0, vmax=1)

# Add outlines for each cell in the maze and gray out the blocked paths

for row in range(solver.rows):

for col in range(solver.cols):

if solver.maze[row][col] == 0:

ax.add\_patch(plt.Rectangle((col - 0.5, row - 0.5), 1, 1, edgecolor='black', facecolor='white', lw=1))

else:

ax.add\_patch(plt.Rectangle((col - 0.5, row - 0.5), 1, 1, edgecolor='black', facecolor='grey', lw=1))

# Create a circle to represent the rat

rat\_circle = plt.Circle((0, 0), 0.3, color='red', fill=True)

ax.add\_artist(rat\_circle)

def update(frame):

# Update the rat's position based on the path

row, col = solver.path[frame]

rat\_circle.center = (col, row)

ax.add\_patch(plt.Rectangle((col - 0.5, row - 0.5), 1, 1, edgecolor='black', facecolor='lightgreen', lw=2))

# If this is the last frame, indicate target achieved

if frame == len(solver.path) - 1:

ax.text(0.5, -0.1, 'Target Achieved!', fontsize=14, color='green', ha='center', transform=ax.transAxes)

ax.add\_patch(plt.Rectangle((col - 0.5, row - 0.5), 1, 1, edgecolor='black', facecolor='green', lw=2))

return rat\_circle,

ani = animation.FuncAnimation(fig, update, frames=len(solver.path), interval=500, blit=True, repeat=False)

plt.show()

# Example usage:

maze = [

[1, 0, 0, 0],

[1, 1, 0, 1],

[0, 1, 0, 0],

[1, 1, 1, 1]

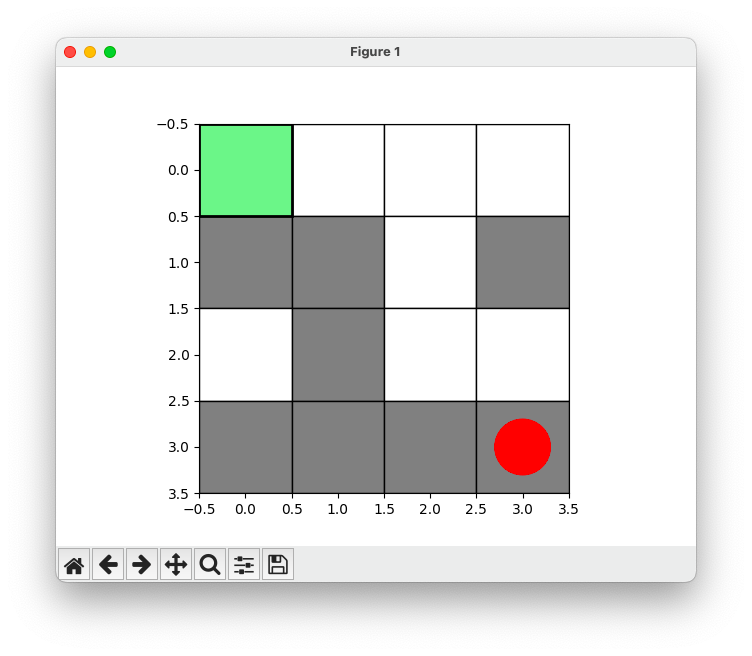
]

solver = MazeSolver(maze)

solver.solve\_maze()

animate\_solution(solver)

OUTPUT:



CODE FOR BFS AND VISUALIZATION:

from collections import deque

import matplotlib.pyplot as plt

import matplotlib.animation as animation

def bfs(maze, start, goal):

rows, cols = len(maze), len(maze[0])

directions = [(-1, 0), (1, 0), (0, -1), (0, 1)] # Up, Down, Left, Right

queue = deque([start])

visited = set()

visited.add(start)

parent = {start: None}

while queue:

current = queue.popleft()

if current == goal:

break

for dr, dc in directions:

nr, nc = current[0] + dr, current[1] + dc

if 0 <= nr < rows and 0 <= nc < cols and maze[nr][nc] == 0 and (nr, nc) not in visited:

queue.append((nr, nc))

visited.add((nr, nc))

parent[(nr, nc)] = current

# Reconstruct the path

path = []

step = goal

while step:

path.append(step)

step = parent.get(step)

path.reverse()

return path

def animate\_solution(maze, path):

fig, ax = plt.subplots()

rows, cols = len(maze), len(maze[0])

# Plot the maze

ax.imshow(maze, cmap="Greys", vmin=0, vmax=1)

# Add outlines for each cell in the maze

for row in range(rows):

for col in range(cols):

ax.add\_patch(plt.Rectangle((col - 0.5, row - 0.5), 1, 1, edgecolor='black', facecolor='none', lw=1))

# Create a circle to represent the rat

rat\_circle = plt.Circle((0, 0), 0.3, color='red', fill=True)

ax.add\_artist(rat\_circle)

def update(frame):

if frame < len(path):

row, col = path[frame]

rat\_circle.center = (col, row)

# Highlight the path cells from start to current cell

for r, c in path[:frame+1]:

ax.add\_patch(plt.Rectangle((c - 0.5, r - 0.5), 1, 1, edgecolor='black', facecolor='lightblue', lw=2))

# Highlight the current cell being visited

if frame > 0:

prev\_row, prev\_col = path[frame - 1]

ax.add\_patch(plt.Rectangle((prev\_col - 0.5, prev\_row - 0.5), 1, 1, edgecolor='black', facecolor='lightgreen', lw=2))

# Indicate the target cell

if frame == len(path) - 1:

ax.add\_patch(plt.Rectangle((col - 0.5, row - 0.5), 1, 1, edgecolor='black', facecolor='green', lw=2))

ax.text(0.5, -0.1, 'Target Achieved!', fontsize=14, color='green', ha='center', transform=ax.transAxes)

# Print the current cell being visited to the terminal

print(f"Step {frame}: Current Position: ({row}, {col})")

return rat\_circle,

ani = animation.FuncAnimation(fig, update, frames=len(path), interval=500, blit=True, repeat=False)

plt.show()

# Example Maze

maze = [

[1, 0, 0, 0],

[1, 1, 0, 1],

[0, 1, 0, 0],

[1, 1, 1, 1]

]

start = (0, 0) # Starting point

goal = (3, 3) # Goal point

path = bfs(maze, start, goal)

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

animate\_solution(maze, path)

A screenshot of a computer game

Description automatically generated