## Indian Institute of Information Technology Surat

****

# Lab Report on

# Artificial Intelligence (CS 701) Practical

**Submitted by**

### [RAHUL KUMAR SINGH] (UI21CS44)

**Course Faculty**

### Dr. Ritesh Kumar

### Mrs. Archana Balmik

## Department of Computer Science and Engineering

## Indian Institute of Information Technology Surat

## Gujarat-394190, India

**Aug-2024**

## Lab No: 8

## Aim:

To develop algorithms for generating a maze and solving the Traveling Salesman Problem using heuristic methods.

## Description:

### Maze

* Utilizes Depth-First Search (DFS) for maze generation.
* Random Maze with random starting cell and finish cell denoted by a random seed.
* Path Creation using random wall removal.
* Enables movement using arrow keys or WASD controls.

### TSP

* Finds the shortest route visiting each city exactly once and returning to the start.
* Implements a heuristic search function to improve efficiency.
* Aims to minimize the total travel distance through smart path selection.

## Code:

**A)Maze**

import pygame

import random

import sys

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

RED = (255, 0, 0)

GREEN = (0, 255, 0)

GRAY = (200, 200, 200)

def generate\_maze(width, height):

maze = [['#'] \* width for \_ in range(height)]

stack = []

start\_x = random.randint(1, width - 2)

start\_y = random.randint(1, height - 2)

maze[start\_y][start\_x] = ' '

stack.append((start\_x, start\_y))

while stack:

x, y = stack[-1]

neighbors = []

for dx, dy in [(2, 0), (-2, 0), (0, 2), (0, -2)]:

nx, ny = x + dx, y + dy

if 0 < nx < width - 1 and 0 < ny < height - 1 and maze[ny][nx] == '#':

neighbors.append((nx, ny))

if neighbors:

nx, ny = random.choice(neighbors)

maze[ny][nx] = ' '

maze[y + (ny - y) // 2][x + (nx - x) // 2] = ' '

stack.append((nx, ny))

else:

stack.pop()

start\_pos = (start\_x, start\_y)

end\_pos = (random.randint(1, width - 2), random.randint(1, height - 2))

while maze[end\_pos[1]][end\_pos[0]] != ' ':

end\_pos = (random.randint(1, width - 2), random.randint(1, height - 2))

maze[start\_pos[1]][start\_pos[0]] = 'S'

maze[end\_pos[1]][end\_pos[0]] = 'E'

return maze, start\_pos, end\_pos

def draw\_maze(screen, maze, player\_pos, end\_pos):

screen.fill(GRAY)

for y, row in enumerate(maze):

for x, cell in enumerate(row):

if cell == '#':

pygame.draw.rect(screen, BLACK, (x \* 20, y \* 20, 20, 20))

elif cell == 'S':

pygame.draw.rect(screen, GREEN, (x \* 20, y \* 20, 20, 20))

elif cell == 'E':

pygame.draw.rect(screen, (0, 0, 255), (x \* 20, y \* 20, 20, 20))

else:

pygame.draw.rect(screen, WHITE, (x \* 20, y \* 20, 20, 20))

player\_x, player\_y = player\_pos

pygame.draw.rect(screen, RED, (player\_x \* 20, player\_y \* 20, 20, 20))

def main\_menu(screen):

font = pygame.font.Font(None, 48)

title\_text = font.render("Maze Game", True, BLACK)

play\_text = font.render("Play", True, BLACK)

quit\_text = font.render("Quit", True, BLACK)

while True:

screen.fill(WHITE)

screen.blit(title\_text, (100, 50))

screen.blit(play\_text, (150, 150))

screen.blit(quit\_text, (150, 250))

pygame.display.flip()

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

if event.type == pygame.MOUSEBUTTONDOWN:

mouse\_pos = pygame.mouse.get\_pos()

if 150 <= mouse\_pos[0] <= 250:

if 150 <= mouse\_pos[1] <= 200:

return "play"

elif 250 <= mouse\_pos[1] <= 300:

pygame.quit()

sys.exit()

def size\_query(screen):

font = pygame.font.Font(None, 36)

input\_box = pygame.Rect(100, 150, 140, 32)

color\_inactive = pygame.Color('lightskyblue3')

color\_active = pygame.Color('dodgerblue2')

color = color\_inactive

active = False

size\_text = ''

prompt\_text = font.render("Enter Maze Size (width height):", True, BLACK)

while True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

if event.type == pygame.MOUSEBUTTONDOWN:

if input\_box.collidepoint(event.pos):

active = not active

else:

active = False

color = color\_active if active else color\_inactive

if event.type == pygame.KEYDOWN:

if active:

if event.key == pygame.K\_RETURN:

try:

width, height = map(int, size\_text.split())

width+=(width+1)%2

height+=(height+1)%2

if width % 2 == 0 or height % 2 == 0:

raise ValueError

return width, height

except ValueError:

size\_text = ''

elif event.key == pygame.K\_BACKSPACE:

size\_text = size\_text[:-1]

else:

size\_text += event.unicode

screen.fill(WHITE)

screen.blit(prompt\_text, (50, 50))

txt\_surface = font.render(size\_text, True, color)

width\_input = max(200, txt\_surface.get\_width()+10)

input\_box.w = width\_input

screen.blit(txt\_surface, (input\_box.x+5, input\_box.y+5))

pygame.draw.rect(screen, color, input\_box, 2)

pygame.display.flip()

def game\_loop(screen, width, height):

maze, start\_pos, end\_pos = generate\_maze(width, height)

player\_pos = list(start\_pos)

while True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

keys = pygame.key.get\_pressed()

if keys[pygame.K\_w] and maze[player\_pos[1] - 1][player\_pos[0]] in (' ', 'E'):

player\_pos[1] -= 1

if keys[pygame.K\_s] and maze[player\_pos[1] + 1][player\_pos[0]] in (' ', 'E'):

player\_pos[1] += 1

if keys[pygame.K\_a] and maze[player\_pos[1]][player\_pos[0] - 1] in (' ', 'E'):

player\_pos[0] -= 1

if keys[pygame.K\_d] and maze[player\_pos[1]][player\_pos[0] + 1] in (' ', 'E'):

player\_pos[0] += 1

pygame.time.delay(100)

if tuple(player\_pos) == end\_pos:

return

draw\_maze(screen, maze, player\_pos, end\_pos)

pygame.display.flip()

def main():

pygame.init()

screen = pygame.display.set\_mode((1800, 1000))

pygame.display.set\_caption("Maze Game")

while True:

choice = main\_menu(screen)

if choice == "play":

width, height = size\_query(screen)

game\_loop(screen, width, height)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**B)TSP**

## import numpy as np

## class TSP:

## def \_\_init\_\_(self, distance\_matrix):

## self.distance\_matrix = distance\_matrix

## self.num\_cities = len(distance\_matrix)

## def nearest\_neighbor(self, start\_city):

## visited = [False] \* self.num\_cities

## visited[start\_city] = True

## tour = [start\_city]

## total\_distance = 0

## current\_city = start\_city

## for \_ in range(self.num\_cities - 1):

## nearest\_city = None

## nearest\_distance = float('inf')

## for city in range(self.num\_cities):

## if not visited[city] and self.distance\_matrix[current\_city][city] < nearest\_distance:

## nearest\_city = city

## nearest\_distance = self.distance\_matrix[current\_city][city]

## tour.append(nearest\_city)

## total\_distance += nearest\_distance

## visited[nearest\_city] = True

## current\_city = nearest\_city

## total\_distance += self.distance\_matrix[current\_city][start\_city]

## tour.append(start\_city)

## return tour, total\_distance

## 

## def main():

## n = int(input("Enter the count of nodes: "))

## distance\_matrix = (np.random.rand(n,n)\*100).astype(int)

## print("Distance Matrix", distance\_matrix)

## tsp = TSP(distance\_matrix)

## start\_city = 0

## tour, total\_distance = tsp.nearest\_neighbor(start\_city)

## print("Tour:", tour)

## print("Total Distance:", total\_distance)

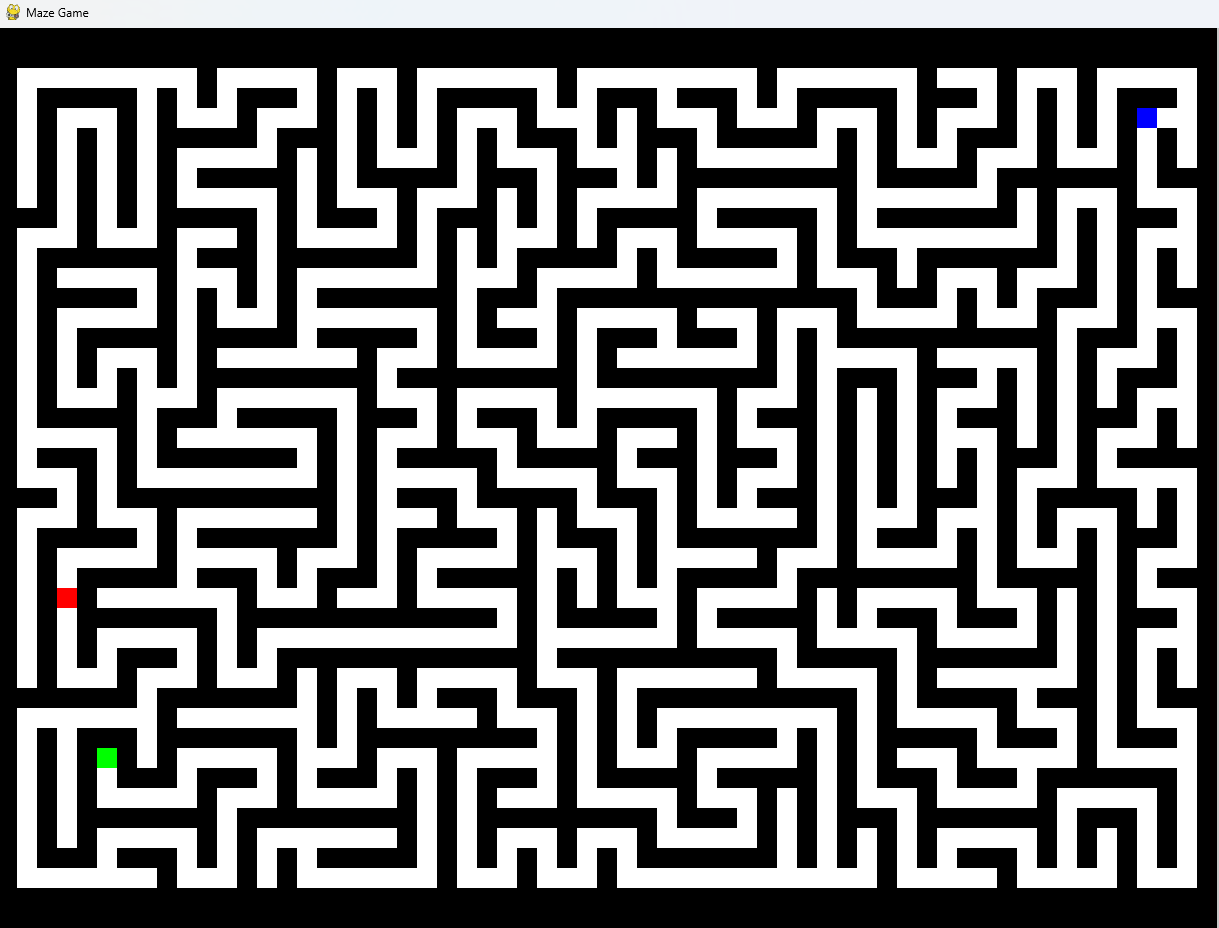
## 

## if \_\_name\_\_ == "\_\_main\_\_":

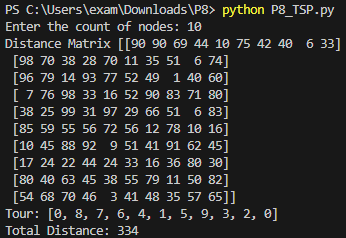
## main()

## Output:

**A)Maze (Green: Start || Red: Player || Blue: Finish)**



**B)TSP**

****

## Conclusion:

* Recent algorithms, such as Genetic Algorithms, Ant Colony Optimization, and Simulated Annealing, improve routing and optimization capabilities.
* Heuristic approaches like A\*, Greedy Best-First Search, and Iterative Deepening A\* offer effective solutions for pathfinding.
* Integrating diverse strategies can lead to better performance depending on problem complexity.