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
```

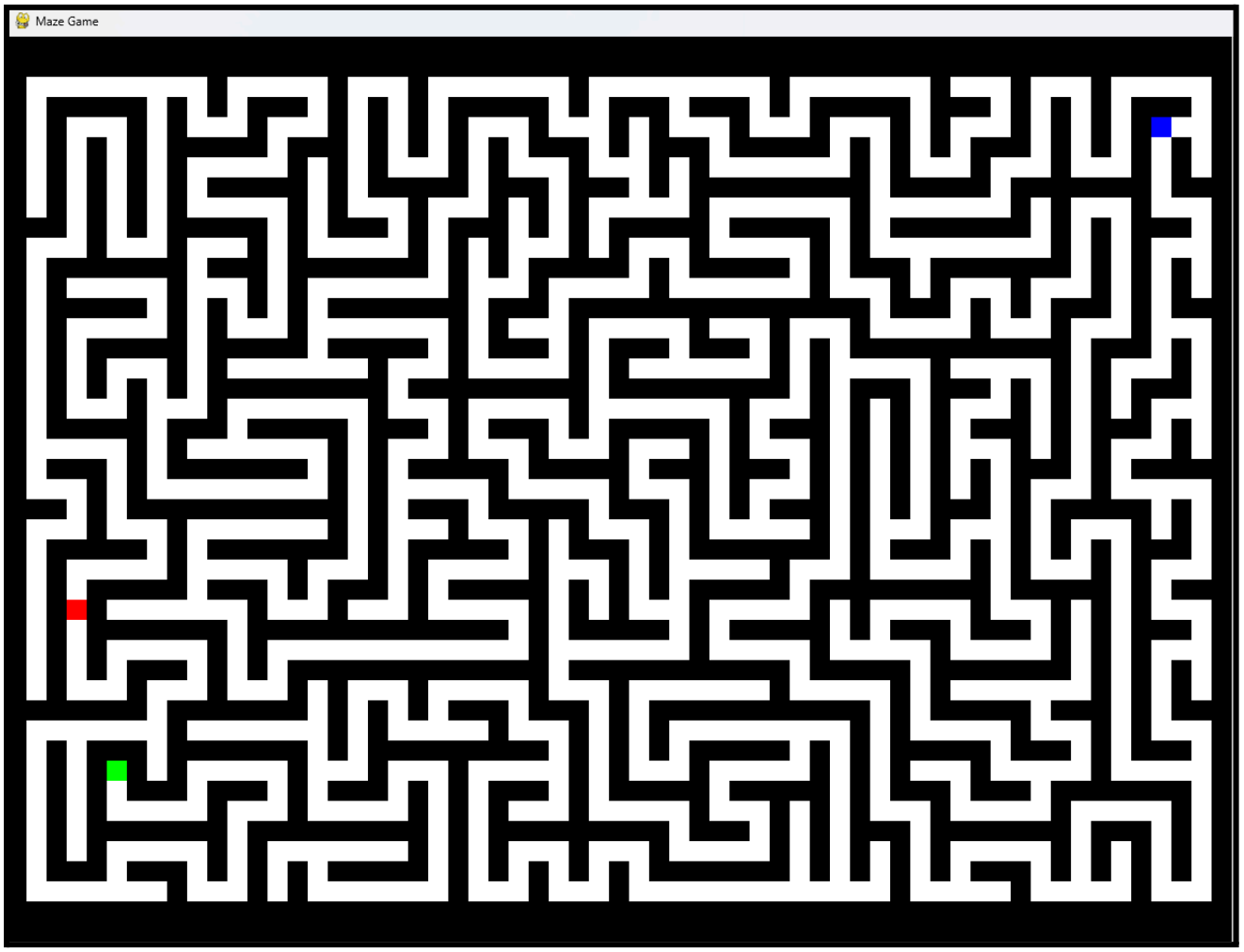
```
        total_distance += self.distance_matrix[current_city][start_city]
        tour.append(start_city)
        return tour, total_distance

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

```
PS C:\Users\exam\Downloads\P8> python P8_TSP.py
Enter the count of nodes: 10
Distance Matrix [[90 90 69 44 10 75 42 40  6 33]
 [98 70 38 28 70 11 35 51  6 74]
 [96 79 14 93 77 52 49  1 40 60]
 [ 7 76 98 33 16 52 90 83 71 80]
 [38 25 99 31 97 29 66 51  6 83]
 [85 59 55 56 72 56 12 78 10 16]
 [10 45 88 92  9 51 41 91 62 45]
 [17 24 22 44 24 33 16 36 80 30]
 [80 40 63 45 38 55 79 11 50 82]
 [54 68 70 46  3 41 48 35 57 65]]
Tour: [0, 8, 7, 6, 4, 1, 5, 9, 3, 2, 0]
Total Distance: 334
```

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.