CHANAKYA UNIVERSITY

SCHOOL OF ENGINEERING

ASSIGNMENT: 1

(WEEK-1 REPORT)

ASSIGNMENT TITLE:

BOT-BRAIN- WAYFINDING AGENT FOR CHANAKYA UNIVERSITY

SUBMITTED BY:

HARSHITHA P

REGISTER NO: 24UG00551

SEMISTER III

SUBJECT: INTRODUCTION TO AIML

SECTION: B

ABSTRACT:

Bot-Brain is a digital campus guide for Chanakya University that helps students to reach their destinations easily. It uses AI-based algorithms and techniques like BFS, DFS, A* and UCS to find the shortest and safest walk routes between any two locations with a detailed digital map of important places like academic blocks, library, hostel blocks, food court, and sports area. This makes navigation easy**.** Students can ask natural questions like "How do I get food court from the admin block?". Its friendly interface and it allows students to reach their destinations quickly, confidently and without any confusion, and it makes university travels easy and trouble-free.

INTRODUCTION:

Chanakya University is large campus it can be confusing for new, old students and visitors. To help them find their way easily, the project “Bot-Brain” is being created. It is a smart system that uses AI and a digital map of the campus. The project aims to build a detailed map by connecting major buildings and measuring walking distances. It will offer a simple text-based interface where users can type their starting point and destination, choose a navigation method, and get step-by-step directions.

Problem statement:

The lack of clear maps and sufficient guidance makes it difficult for students and visitors to navigate the campus. Finding academic blocks, library, and other necessary locations in the campus can become difficult and time consuming. so, there is a need for an , AI-driven navigation solution that makes it easier for students and visitors to explore campus and makes their time there much better.

Objectives:

The goal of the system is to find the best walking paths between any two places on campus, making navigation faster and more efficient. It will have a full digital map that shows all the main university buildings, like academic blocks, library, hostel blocks, mart, food court. The system will also support natural language queries, which will make it easy for users to get information and interact with each other. such as when it is open and what facilities are available. In general, the goal of the solution is to make it easy, quick, and stress-free for students, faculty, and visitors to get around campus.

- Create a complete digital map
- Experiment with various search methods
- Allow students to request routes easily:
- Display significant building information:
- Test and compare the search techniques:

Scope of the project:

- Campus Wide Coverage: Covers all major academic blocks and other spots of the campus it ensure anyone can easily navigate easiest routes across the campus.
- Flexible use of Algorithms: It Utilizes various search strategies to acquire optimal routes between campus points.
- Detailed explanation of buildings: Displays functional details for every locations in the campus and opening times, facilities provided, and emergency contact numbers.
- Text-Based Interface: Facilitates the straightforward querying of paths and building details through an easy text based interface.
- In future it could be extended to include the emergency navigation help with parking notification.

Documents Required:

- **Identify Campus Locations:**

  List the important places on campus that students frequently visit, such as Academic Block, the Library, Admin block, food court, nurse's room, Auditorium, sports ground hostel blocks, mart, main gate, exit gate.

- **Map connections between the buildings:**

  Estimate the walking paths that directly connected to other buildings These distances can be estimated from an actual map of the campus and measured via Google Maps (satellite, measure)

- **Gathering the building details:**

  Collect the information of every location in the campus and it includes opening and closing times like food court and library open and closing timings. available facilities in the campus and contact numbers.

- **Algorithms and data structure**

  Build a graph model where all the buildings are points or nodes and path are edges. This structure will allow the algorithms to efficiently find and compare the best routes in the all possible ways to reach the destination.

These documents are crucial for developing a user-friendly campus wayfinding system, covering physical mappings, user interaction design, and the algorithm verification.

Tools & technology list:

- **Python:**

  The main programming language is used to build the smart navigation system because it is powerful and easy to use, especially for AI tasks.

- **Tkinter / PySimpleGUI / Flask:**

  These are tools for making the user interface. Tkinter and PySimpleGUI help create simple desktop apps quickly, and Flask can be used to create a simple website for the system.

- **GitHub:**

  A platform to keep track of all the code changes, manage the project, and work safely with others.

- **Google Maps API:**

  An optional tool that provides real satellite images of the campus to help make the digital map more accurate.

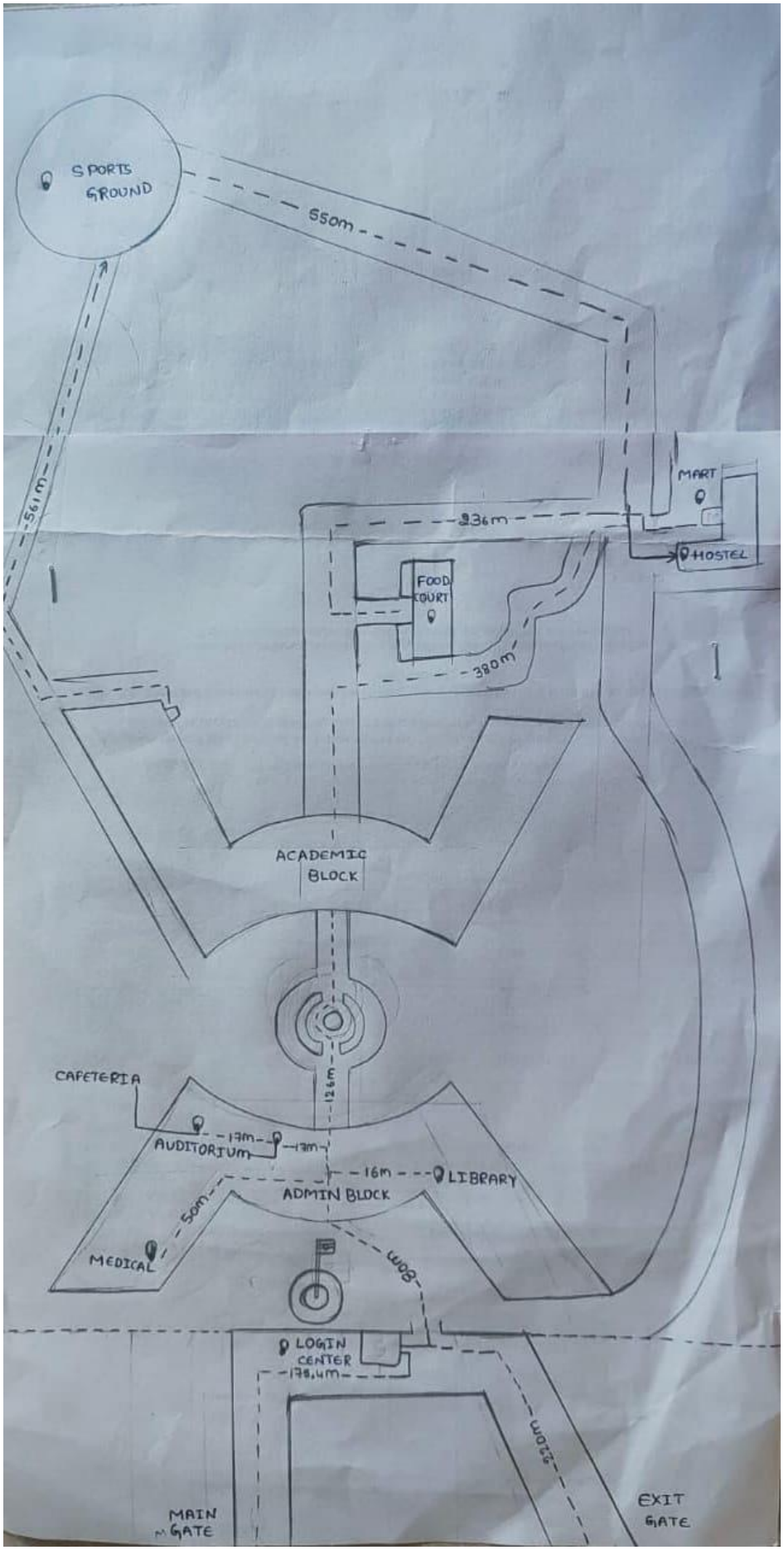
- **PowerPoint / Word:**

  Programs used for making diagrams, writing reports, and creating presentations to explain the project clearly.

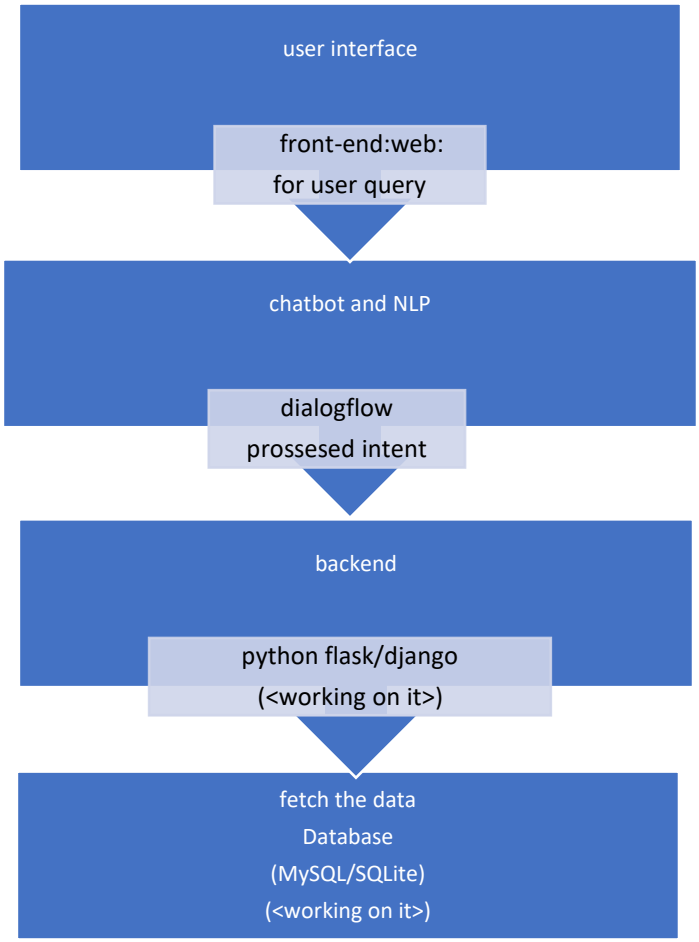
CAMPUS LAYOUT:

Main locations in campus:

- Main gate
- Login centre
- Admin block
- Library
- Cafeteria
- Auditorium
- Academic block
- Food court
- Sports ground
- Hostel block
- Medical room
- Exit gate


SPORTS GROUND
550m
561m
236m
MART
HOSTEL
FOOD COURT
380m
ACADEMIC BLOCK
CAFETERIA
126m
AUDITORIUM
17m
17m
16m
LIBRARY
ADMIN BLOCK
50m
MEDICAL
80m
LOGIN CENTER
178.4m
120m
MAIN GATE
EXIT GATE

Working Database Connection:



CODE

```
# main.py — Wayfinding Agent for Chanakya University

import math
import os
import tkinter as tk
from tkinter import ttk
from queue import PriorityQueue
from PIL import Image, ImageTk, ImageDraw

# ---------------- Coordinates ----------------
location_coords = {
    "Admin Block": (310, 250),
    "Library": (350, 250),
    "Academic Block": (310, 200),
    "Cafeteria": (275, 250),
    "Auditorium": (290, 245),
    "Medical Room": (250, 270),
    "Hostel": (465, 90),
    "Mini Mart": (455, 80),
    "Food Court": (340, 140),
    "Main Gate": (310, 380),
    "Exit Gate": (550, 380),
    "Security Entrance": (350, 310),
    "Sports": (215, 25),
}

# ---------------- Campus corridors ----------------
edge_paths = {
    ("Admin Block", "Academic Block"): [(310, 250), (310, 230), (310, 200)],
```

```
    ("Admin Block", "Library"): [(310, 250), (330, 250), (350, 250)],
    ("Admin Block", "Cafeteria"): [(310, 250), (295, 250), (275, 250)],
    ("Admin Block", "Auditorium"): [(310, 250), (300, 248), (290, 245)],
    ("Admin Block", "Medical Room"): [(310, 250), (290, 258), (270, 265), (250, 270)],
    ("Security Entrance", "Admin Block"): [(350, 310), (335, 290), (320, 270), (310, 250)],
    ("Security Entrance", "Academic Block"): [(350, 310), (342, 285), (328, 240), (310, 200)],
    ("Security Entrance", "Food Court"): [(350, 310), (346, 270), (345, 220), (342, 180), (340,
140)],
    ("Security Entrance", "Main Gate"): [(350, 310), (332, 345), (310, 380)],
    ("Security Entrance", "Exit Gate"): [(350, 310), (420, 350), (550, 380)],
    ("Academic Block", "Food Court"): [(310, 200), (320, 180), (330, 160), (340, 140)],
    ("Food Court", "Hostel"): [(340, 140), (360, 125), (400, 110), (435, 100), (465, 90)],
    ("Hostel", "Mini Mart"): [(465, 90), (460, 85), (455, 80)],
    ("Food Court", "Sports"): [(340, 140), (300, 100), (260, 60), (230, 40), (215, 25)],
}

# ---------------- Build adjacency strictly from corridors ----------------
def polyline_length(points):
    return sum(
        math.hypot(points[i+1][0]-points[i][0], points[i+1][1]-points[i][1])
        for i in range(len(points)-1)
    )

def build_adjacency(edge_paths):
    adj = {name: {} for name in location_coords}
    for (a, b), pts in edge_paths.items():
        L = polyline_length(pts)
        adj[a][b] = L
        adj[b][a] = L
    return adj
```

```
CORRIDOR_ADJ = build_adjacency(edge_paths)


# ---------------- Heuristic and A* ----------------
def build_heuristics(coords, goal):
    gx, gy = coords[goal]
    return {name: math.hypot(x - gx, y - gy) for name, (x, y) in coords.items()}


def a_star(graph, heur, start, goal):
    frontier = PriorityQueue()
    frontier.put((0.0, start))
    came = {start: None}
    g = {start: 0.0}
    while not frontier.empty():
        _, u = frontier.get()
        if u == goal:
            break
        for v, d in graph.get(u, {}).items():
            nv = g[u] + d
            if v not in g or nv < g[v]:
                g[v] = nv
                frontier.put((nv + heur.get(v, 0.0), v))
                came[v] = u
    if goal not in came:
        return None, None
    path = []
    x = goal
    while x is not None:
        path.append(x)
        x = came[x]
    path.reverse()
```

```
    return path, g[path[-1]]


# ---------- Contacts and Timings ----------
contacts = {
    "Admin Block": {
        "Admission": [
            {"name": "Thanushree", "phone": "08031233133"},
            {"name": "Sri Vijay Kumar", "phone": "08031233103"}
        ],
        "Registrar": [
            {"name": "Sri Gautam", "phone": "08031233104"},
            {"name": "Sri Dhanashri", "phone": "08031233101"}
        ],
        "Communication": {"name": "Sri Chandrashekar", "phone": "08031233107"},
        "Finance": {"name": "Sri Sudheerda K.M", "phone": "08031233102"},
        "IT Support": {"name": "Sri Sachin Goni", "phone": "08031233109"}
    },
    "Academic Block": {
        "VC Office": [
            {"name": "Shilparanganathra", "phone": "08031233122"},
            {"name": "Ashwin Kumar", "phone": "Contact VC Office"},
            {"name": "Subhant T. Joshi", "phone": "08031233104"},
            {"name": "Dr. Padmavathi B.S", "phone": "Contact VC Office"},
            {"name": "Dr. Vineeth Paleri", "phone": "Contact VC Office"}
        ]
    },
    "Library": {"Librarian": {"name": "Bharathkumar V", "phone": "8861775721"}},
    "Medical Room": {"Doctor": {"name": "Dr. Anjana", "phone": "08031233103"}},
    "Sports": {"Coach": {"name": "Sri Hemanth", "phone": "9449141869"}},
    "Hostel": {
```

```
        "Girls": [
            {"name": "Jayashree", "phone": "09513228510"},
            {"name": "Arathi", "phone": "09513228509"}
        ],
        "Boys": {"name": "Ullas Kallur", "phone": "08031233100"}
    }
}

location_timings = {
    "Admin Block": {"start": "9:30", "end": "17:30"},
    "Academic Block": {"start": "9:30", "end": "17:30"},
    "Library": {"start": "8:30", "end": "21:30"},
    "Cafeteria": {"start": "9:30", "end": "17:30"},
    "Auditorium": {"start": "9:30", "end": "17:30"},
    "Food Court": {
        "breakfast": {"start": "7:30", "end": "9:15"},
        "lunch": {"start": "12:30", "end": "14:15"},
        "dinner": {"start": "19:30", "end": "22:00"}
    },
    "Main Gate": {"start": "7:00", "end": "20:00"},
    "Security Entrance": {"start": "7:00", "end": "20:00"},
    "Hostel": {"start": "24 Hours", "end": ""}
}

# ---------------- GUI ----------------
root = tk.Tk()
root.title("Wayfinding Agent for Chanakya University")
root.geometry("940x740")

frame = ttk.Frame(root)
```

```
frame.pack(fill="both", expand=True)


tk.Label(frame, text="Enter your current location:").grid(row=0, column=0, padx=10,
pady=6, sticky="e")

start_cb = ttk.Combobox(frame, values=sorted(location_coords.keys()), width=28)

start_cb.grid(row=0, column=1, padx=10, pady=6)


tk.Label(frame, text="Enter your target location:").grid(row=1, column=0, padx=10, pady=6,
sticky="e")

end_cb = ttk.Combobox(frame, values=sorted(location_coords.keys()), width=28)

end_cb.grid(row=1, column=1, padx=10, pady=6)


out = tk.Text(frame, height=20, width=78)

out.grid(row=2, column=0, columnspan=2, padx=10, pady=6)


# ---------- Map loading ----------

def load_map():

    candidates = [

        "C:\\Users\\Vijay\\OneDrive - Chanakya University\\Desktop\\collage\\WhatsApp
Image 2025-09-19 at 21.51.24_c4dd2f93.jpg",  # Collage map photo

        "image.jpg",

        "campus_map.jpg",

        "WhatsApp-Image-2025-09-19-at-13.03.49_6ae8d95e.jpg",

        "WhatsApp-Image-2025-09-19-at-21.51.32_aa402119.jpg",

    ]

    for name in candidates:

        if os.path.exists(name):

            return Image.open(name).resize((600, 400))

    im = Image.new("RGB", (600, 400), "white")

    d = ImageDraw.Draw(im)

    d.text((10, 10), "Campus Map Missing", fill="black")
```

```
    return im

img = load_map()
photo = ImageTk.PhotoImage(img)

canvas = tk.Canvas(frame, width=600, height=400, bg="white")
canvas.grid(row=4, column=0, columnspan=2, padx=10, pady=10)
canvas.image = photo
canvas.create_image(0, 0, anchor="nw", image=photo)

# ---------- Draw corridors ----------
PATH_COLOR = "#17a45b"
BASE_COLOR = "#9aa3a8"

def draw_base_graph():
    canvas.delete("base_edge")
    for (a, b), pts in edge_paths.items():
        for i in range(len(pts)-1):
            x1, y1 = pts[i]
            x2, y2 = pts[i+1]
            canvas.create_line(x1, y1, x2, y2, fill=BASE_COLOR, width=2,
tags=("base_edge",))




for name, (x, y) in location_coords.items():
    canvas.create_oval(x-3, y-3, x+3, y+3, fill="#0e7", outline="")
    canvas.create_text(x+8, y-10, text=name, anchor="w", font=("Arial", 8), fill="#024")

def all_missing_polylines(path):
    missing = []
```

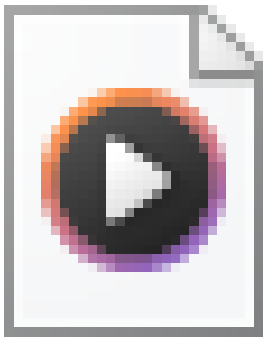
```
    for a, b in zip(path, path[1:]):
        if not (edge_paths.get((a, b)) or edge_paths.get((b, a))):
            missing.append(f"{a} → {b}")
    return missing


def draw_segment(points, color=PATH_COLOR):
    for i in range(len(points)-1):
        x1, y1 = points[i]
        x2, y2 = points[i+1]
        canvas.create_line(x1, y1, x2, y2, fill=color, width=6, capstyle=tk.ROUND,
joinstyle=tk.ROUND, tags=("path_line",))


def draw_path(path):
    canvas.delete("path_line")
    for a, b in zip(path, path[1:]):
        pts = edge_paths.get((a, b)) or edge_paths.get((b, a))
        draw_segment(pts, PATH_COLOR)
    sx, sy = location_coords[path[0]]
    gx, gy = location_coords[path[-1]]
    canvas.create_oval(sx-5, sy-5, sx+5, sy+5, fill="#1e88e5", outline="", tags=("path_line",))
    canvas.create_oval(gx-7, gy-7, gx+7, gy+7, outline=PATH_COLOR, width=3, fill="",
tags=("path_line",))
    canvas.tag_raise("path_line")


def find_path():
    start = start_cb.get().strip()
    end = end_cb.get().strip()
    out.delete("1.0", tk.END)
    if start not in CORRIDOR_ADJ or end not in CORRIDOR_ADJ:
        out.insert(tk.END, "Invalid start or destination.\n")
        return
```

```
    if start == end:
        out.insert(tk.END, "Start and destination are the same.\n")
        return
    heur = build_heuristics(location_coords, end)
    path, cost = a_star(CORRIDOR_ADJ, heur, start, end)
    if not path:
        out.insert(tk.END, "No path found.\n")
        return
    missing = all_missing_polylines(path)
    if missing:
        out.insert(tk.END, "Missing corridor polylines for: " + ", ".join(missing) + "\n")
        return
    minutes = round(cost * 0.02, 1)
    out.insert(tk.END, f"Path: {' → '.join(path)}\n")
    out.insert(tk.END, f"Distance (units): {int(cost)}\n")
    out.insert(tk.END, f"Estimated Time: {minutes} minutes\n")
    draw_path(path)

def show_details():
    target = end_cb.get().strip()
    out.delete("1.0", tk.END)
    # Timings
    timing = location_timings.get(target)
    if timing:
        out.insert(tk.END, f"{target} Timings:\n")
        if isinstance(timing, dict):
            for key, val in timing.items():
                if isinstance(val, dict):
                    out.insert(tk.END, f"{key.capitalize()}: {val.get('start','')} - {val.get('end','')}\n")
                else:
```

```
                out.insert(tk.END, f"{key}: {val}\n")
        out.insert(tk.END, "\n")
    # Contacts
    info = contacts.get(target)
    if info:
        out.insert(tk.END, "Contacts:\n")
        for role, val in info.items():
            if isinstance(val, list):
                out.insert(tk.END, f"{role}:\n")
                for person in val:
                    out.insert(tk.END, f"  Name: {person['name']}, Phone: {person['phone']}\n")
            elif isinstance(val, dict):
                if "name" in val:
                    out.insert(tk.END, f"{role}:\n  Name: {val['name']}, Phone: {val['phone']}\n")
                else:
                    out.insert(tk.END, f"{role}:\n")
                    for subrole, subval in val.items():
                        if isinstance(subval, list):
                            out.insert(tk.END, f"  {subrole}:\n")
                            for person in subval:
                                out.insert(tk.END, f"    Name: {person['name']}, Phone:
{person['phone']}\n")
                        elif isinstance(subval, dict):
                            out.insert(tk.END, f"  {subrole}:\n    Name: {subval.get('name','')}, Phone:
{subval.get('phone','')}\n")
        out.insert(tk.END, "\n")
    else:
        out.insert(tk.END, "No contact details available.\n")


tk.Button(frame, text="Find Path", command=find_path).grid(row=3, column=0, pady=6)
```

```
tk.Button(frame, text="Show Location Details", command=show_details).grid(row=3,
column=1, pady=6)


# Defaults for testing
start_cb.set("Food Court")
end_cb.set("Hostel")


root.mainloop()
```

OUTPUT (VIDEO DEMO)



Screen Recording 2025-09-22 114852.m