CHANAKYA UNIVERSITY

SCHOOL OF ENGINEERING

ASSIGNMENT TITLE: CHANAKYA UNIVERSITY DIGITAL PATH FINDER

ASSIGNMENT -1

SUBMITTED BY:

VARSHITHA.T

REGISTER NO:24UG00549

SEMESTER III (ODD)

SUBJECT: INTRODUCTION TO AIML

SECTION B

ABSTRACT

The Bot Brain project aims to develop a smart digital navigation assistant for Chanakya University. The system models campus navigation as a weighted graph, where the buildings are modelled as nodes and the paths that connect the buildings are modelled as edges. Bot Brain determines the best routes for users in as little time as possible using classical AI search algorithms such as Breadth-First Search (BFS), Depth-First Search (DFS), Uniform Cost Search (UCS), and A*, with heuristics based on search methodology. In addition to providing the best route, Bot Brain also provides valuable information about the buildings such as facilities, timings, and available services in relation to their use and purpose, thereby improving the overall campus experience. A primary intent of the project is to compare the various search algorithms and allow students to discover for themselves the effect of algorithm choices on performance and efficiency in real world applications. By combining navigation, access to information and analysis of algorithm, Bot Brain provides a tangible AI-based solution that enhances the campus experience, whether it is improving the accessibility, efficiency, or user friendliness of the campus for students and visitors alike.

INTRODUCTION

With its academic, residential, and recreational amenities, a university campus is like a miniature city. Long walking paths, numerous connected buildings, and a dearth of interactive navigation systems make it difficult for new students to navigate such a campus. Conventional approaches, such as static signboards or printed maps, are frequently inadequate.

By using search algorithms to find the best routes and representing the campus as a graph, artificial intelligence offers a solution. Through the use of AI agent design, the Bot Brain project provides contextual information about buildings and models intelligent navigational decision-making. The system makes use of A* for heuristic-driven efficiency, UCS for cost-aware routing, and BFS and DFS for basic search. Beyond the classroom, the project gives students practical knowledge of intelligent agents, search issues, and algorithm evaluation.

PROBLEM STATEMENT

It can be difficult for visitors and new students to find their way around Chanakya University's expansive and strange campus. They struggle with things like not knowing where buildings are, wasting time looking for offices or classrooms, needing help from others to get directions, and not having easy access to building information like services or timings. This results in ineffective orientation, stress, and delays.

An intelligent campus navigation assistant is required to address this issue, guiding users with the shortest routes, walking times, and building details to make campus exploration more efficient and easier for newcomers.

OBJECTIVES

Objectives

1. Graph Model Development: Design a comprehensive graph model representing Chanakya University's campus, encompassing at least 12 key buildings.

2. Algorithm Implementation: Implement and integrate Breadth-First Search (BFS), Depth-First Search (DFS), Uniform Cost Search (UCS), and A* algorithms to facilitate efficient navigation.

3. User Query Support: Enable users to pose queries such as "Find path from Hostel to Library" or "Navigate from Main Gate to Admin Block," and provide accurate responses.

4. Route Computation and Display: Calculate and display the shortest routes between destinations, including distance and estimated walking time.

5. Building and Service Information: Provide supplementary details about campus buildings and services, enhancing user experience.

6. Algorithmic Performance Comparison: Conduct a comparative analysis of the implemented algorithms in terms of:

   - Path optimality

   - Number of nodes explored

   - Computational efficiency

By achieving these objectives, the project aims to develop an intelligent campus navigation system that offers efficient routing, informative building details, and insightful algorithmic comparisons.

SCOPE

Key Features

1. Graph-Based Campus Model: A comprehensive graph representation of the campus, enabling efficient navigation.

2. Implementation of 4 Search Algorithms: BFS, DFS, UCS, and A* algorithms for optimal route-finding.

3. Text-Based Interface: User-friendly interface for navigation queries, such as "Find path from Hostel to Library."

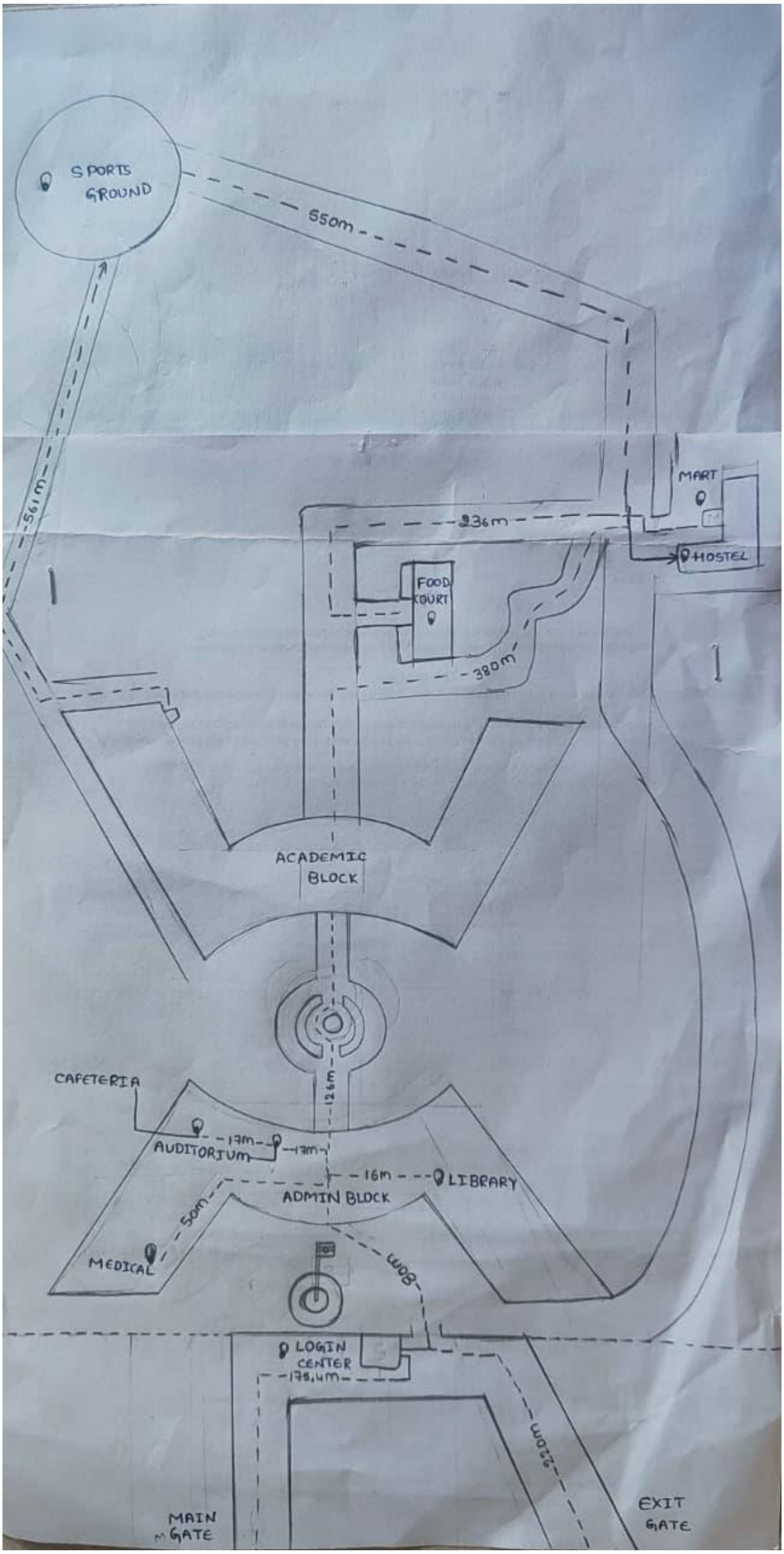
4. Building Information Retrieval: Access to detailed information about campus buildings and services.

5. Algorithm Comparison and Analysis: Comparative evaluation of algorithm performance in terms of path optimality, nodes explored, and efficiency.

These features will be integrated to develop an intelligent campus navigation system, providing users with efficient routing and informative building details.

CAMPUS LAYOUT

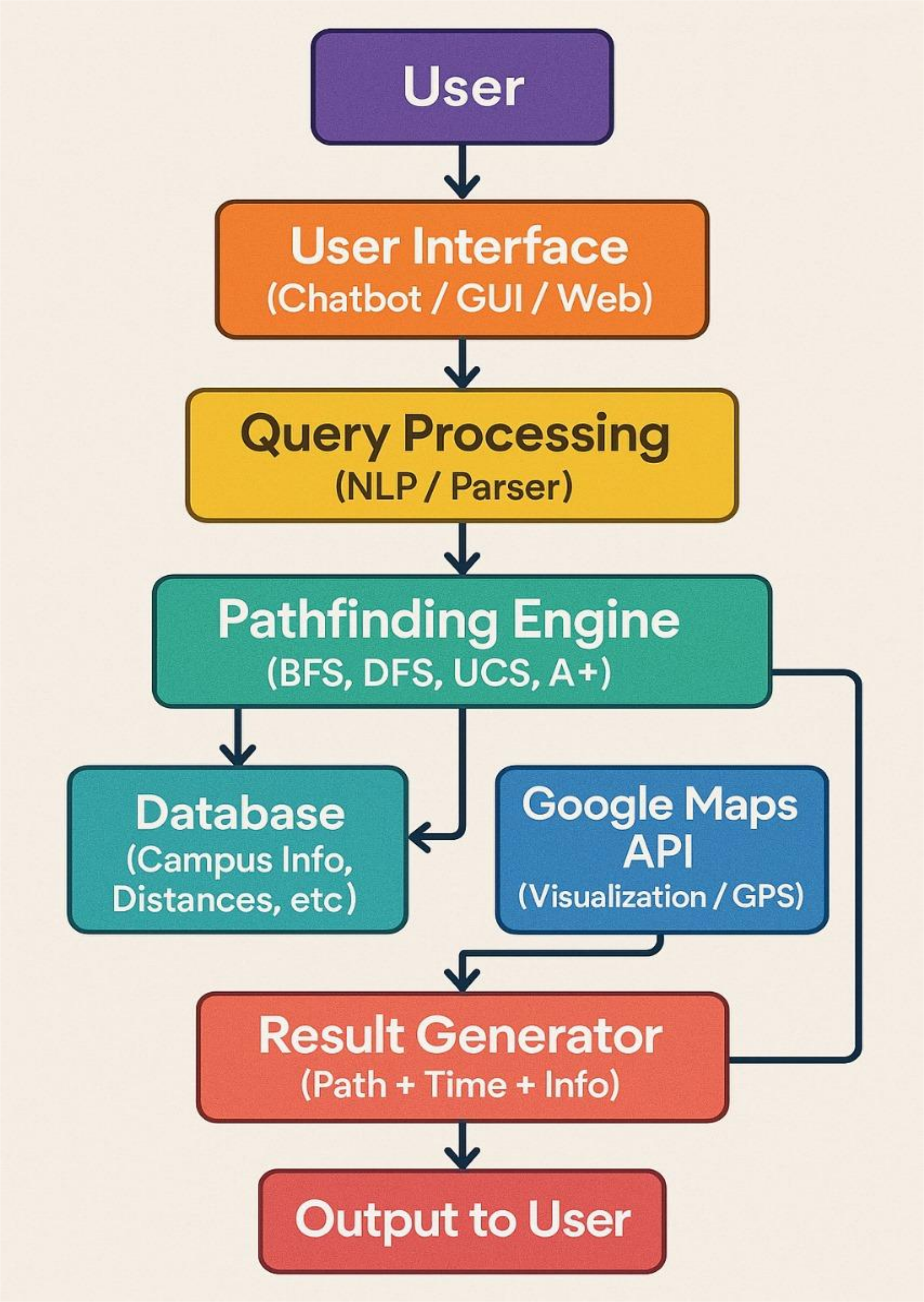
- Main Gate
- Exit Gate
- Admin Block
- Academic Block
- Library
- Canteen
- Hostel
- Mart
- Food Court
- Sports Ground
- Auditorium
- Medical Facility


SPORTS GROUND
550m
561m
MART
236m
HOSTEL
FOOD COURT
380m
ACADEMIC BLOCK
CAFETERIA
126m
17m
17m
AUDITORIUM
16m
LIBRARY
ADMIN BLOCK
50m
MEDICAL
80m
LOGIN CENTER
175.4m
220m
MAIN GATE
EXIT GATE

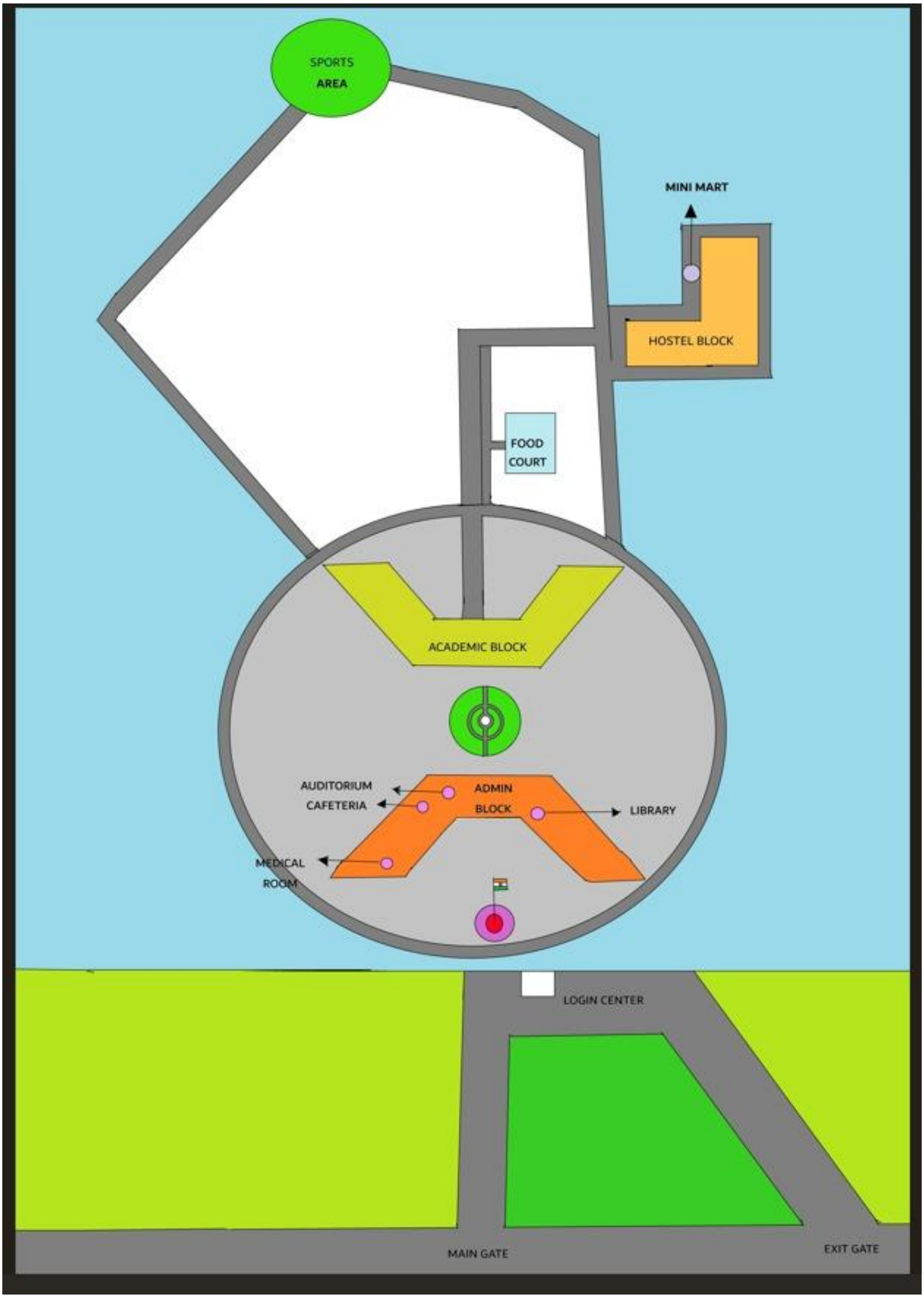
TOOLS & TECHNOLOGIES

- **Python**: Selected as the core programming language for the development of the intelligent agent, implementation of search algorithms, and management of back-end logic. Python is preferred owing to its robustness, simplicity of use, and extensive ecosystem supporting artificial intelligence and computational tasks.
- **Tkinter / Py Simple GUI / Flask**: Frameworks designated for the development of the user interface. Tkinter and PySimpleGUI facilitate the rapid construction of desktop-based graphical interfaces, whereas Flask provides a minimal yet effective platform for web-based deployment.
- **GitHub**: Employed as the version control system to maintain source code integrity, enable collaborative development, streamline code organization, and facilitate systematic peer review processes.
- **Google Maps API**: Integrated to provide satellite-based campus imagery and precise geolocation services. This enables accurate digital mapping of real-world building positions, with optional enhancement for advanced overlay and visualization features.
- **Microsoft PowerPoint / Word**: Utilized for the design and preparation of technical documentation, visual diagrams, system design records, and professional project presentations.

SYSTEM DESIGN AND ARCHITECTURE



Module	Purpose	Feature
User Interface	Engage users via chatbot,GUI,or web	Adaptive UI based on user type
Query processing	NLP based parsing of user queries	Context aware logic
Pathfinding Engine	Implements BFS, DFS, UCS, A* for optimal pathfinding	Custom heuristics
Database	Stores campus data, distances, building info	Dynamic updates from user feedback
Google Maps API	Visualizes routes and syncs with GPS	Campus overlays
Result Generator	Synthesizes path, time, and contextual info	“Why this route?” explanations
Output to User	Presents final results in interactive format	Shareable route cards, voice-guided directions


SPORTS
AREA
MINI MART
HOSTEL BLOCK
FOOD
COURT
ACADEMIC BLOCK
AUDITORIUM
CAFETERIA
ADMIN
BLOCK
LIBRARY
MEDICAL
ROOM
LOGIN CENTER
MAIN GATE
EXIT GATE

FINAL CODE

```
import tkinter as tk

from tkinter import ttk

from queue import PriorityQueue

from collections import deque

from PIL import Image, ImageTk

import math, difflib

from tkinter import messagebox as mb


# ========== Data ==========


distances = {

    "Admin Block": {"Security Entrance": 80,"Library": 16,"Auditorium": 17,"Academic Block": 126,"Cafeteria": 35,"Main Gate": 160,"Exit Gate": 220,"Medical": 80,"Hostel": 506,"Mini Mart": 434,"Food Court": 355,"Sports": 850},

    "Academic Block": {"Security Entrance": 240,"Library": 46,"Auditorium": 150,"Cafeteria": 27,"Main Gate": 210,"Exit Gate": 250,"Medical": 85,"Hostel": 380,"Mini Mart": 410,"Food Court": 192,"Admin Block": 126,"Sports": 561},

    "Library": {"Security Entrance": 100,"Auditorium": 3,"Academic Block": 46,"Cafeteria": 63,"Main Gate": 275,"Exit Gate": 240,"Medical": 75,"Hostel": 450,"Mini Mart": 465,"Food Court": 470,"Admin Block": 16,"Sports": 857},

    "Cafeteria": {"Security Entrance": 130,"Library": 63,"Auditorium": 10,"Academic Block": 27,"Main Gate": 276,"Exit Gate": 270,"Medical": 90,"Hostel": 520,"Mini Mart": 585,"Food Court": 355,"Admin Block": 35,"Sports": 920},

    "Auditorium": {"Security Entrance": 120,"Library": 3,"Academic Block": 150,"Cafeteria": 10,"Main Gate": 286,"Exit Gate": 260,"Medical": 80,"Hostel": 510,"Mini Mart": 574,"Food Court": 344,"Admin Block": 17,"Sports": 930},

    "Food Court": {"Security Entrance": 471,"Library": 470,"Auditorium": 344,"Academic Block": 192,"Cafeteria": 355,"Main Gate": 699,"Exit Gate": 609,"Medical": 405,"Hostel": 816,"Mini Mart": 836,"Admin Block": 355,"Sports": 427},

    "Security Entrance": {"Admin Block": 80,"Academic Block": 240,"Library": 100,"Cafeteria": 130,"Auditorium": 120,"Food Court": 471,"Main Gate": 150,"Exit Gate": 180,"Medical": 90,"Hostel": 500,"Mini Mart": 520,"Sports": 880},

    "Main Gate": {"Admin Block": 160,"Academic Block": 210,"Library": 275,"Cafeteria": 276,"Auditorium": 286,"Food Court": 699,"Security Entrance": 150,"Exit Gate": 100,"Medical": 180,"Hostel": 600,"Mini Mart": 620,"Sports": 900},

    "Exit Gate": {"Admin Block": 220,"Academic Block": 250,"Library": 240,"Cafeteria": 270,"Auditorium": 260,"Food Court": 609,"Security Entrance": 180,"Main Gate": 100,"Medical": 160,"Hostel": 580,"Mini Mart": 600,"Sports": 870},
    "Medical": {"Admin Block": 80,"Academic Block": 85,"Library": 75,"Cafeteria": 90,"Auditorium": 80,"Food Court": 405,"Security Entrance": 90,"Main Gate": 180,"Exit Gate": 160,"Hostel": 490,"Mini Mart": 510,"Sports": 840},

    "Hostel": {"Admin Block": 506,"Academic Block": 380,"Library": 450,"Cafeteria": 520,"Auditorium": 510,"Food Court": 816,"Security Entrance": 500,"Main Gate": 600,"Exit Gate": 580,"Medical": 490,"Mini Mart": 200,"Sports": 300},

    "Mini Mart": {"Admin Block": 434,"Academic Block": 410,"Library": 465,"Cafeteria": 585,"Auditorium": 574,"Food Court": 836,"Security Entrance": 520,"Main Gate": 620,"Exit Gate": 600,"Medical": 510,"Hostel": 200,"Sports": 320},

    "Sports": {"Admin Block": 850,"Academic Block": 561,"Library": 857,"Cafeteria": 920,"Auditorium": 930,"Food Court": 427,"Security Entrance": 880,"Main Gate": 900,"Exit Gate": 870,"Medical": 840,"Hostel": 300,"Mini Mart": 320}

}


contacts = {

    "Admin": {

        "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": {

        "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 Paleni", "phone": "Contact VC Office"}

        ]

    },

    "Library": {"name": "Bharathkumar V", "phone": "8861775721"},

    "Medical Center": {"name": "Dr. Anjana", "phone": "08031233103"},

    "Sports & P.E": {"name": "Sri Hemanth", "phone": "9449141869"},
    "Hostel": {"Girls": {"name": "Jayshree", "phone": "9513228510"}, "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": "9:30", "end": "17:30"},
    "Cafeteria": {"start": "9:30", "end": "17:30"},
    "Auditorium": {"start": "9:30", "end": "17:30"},
    "Food Court": {"breakfast": {"start": "7:30", "end": "10:30"}, "lunch": {"start": "12:30", "end": "14:30"}, "dinner": {"start": "19:30", "end": "22:30"}},
    "Main Gate": {"start": "7:00", "end": "20:00"},
    "Security Entrance": {"start": "7:00", "end": "20:00"},
    "Exit Gate": {"start": "7:00", "end": "20:00"},
    "Medical": {"start": "9:30", "end": "17:30"},
    "Hostel": {"start": "0:00", "end": "23:59"},
    "Mini Mart": {"start": "9:00", "end": "21:00"},
    "Sports": {"start": "6:00", "end": "21:00"}
}

NODE_COORDS = {
    "Main Gate": (420, 1100),
    "Exit Gate": (750, 1100),
    "Security Entrance": (445, 870),
    "Admin Block": (398, 695),
    "Academic Block": (385, 567),
    "Library": (500, 745),
    "Auditorium": (370, 705),
    "Cafeteria": (350, 720),
    "Food Court": (450, 377),
    "Medical": (308, 750),
    "Mini Mart": (620, 205),
    "Hostel": (560, 277),
    "Sports": (270, 55)
}

MAP_PATH = "chanakya_map.jpg"

qa_pairs = {
    "How to get admission?": "Contact the Admissions Office. Do you want the phone number? (Yes/No)",
    "Who is the Registrar?": "The Registrar is available in Admin Block. Do you want contact info? (Yes/No)",
    "Whom to contact in Finance office?": "Visit Finance office in Admin Block. Do you want phone? (Yes/No)",
    "Who is the Librarian?": "The Librarian is available at Library. Need phone number? (Yes/No)",
    "Library timings?": "Library is open: " + location_timings.get("Library", {}).get("start", "?") + " to " + location_timings.get("Library", {}).get("end", "?"),
    "Medical Help?": "Medical Center is available. Do you want the doctor’s contact? (Yes/No)",
    "Who handles IT support?": "IT Support is in Admin Block. Do you want phone? (Yes/No)",
    "Who manages Sports activities?": "Sports in-charge is available. Want phone contact? (Yes/No)",
    "Who is the Girls Hostel warden?": "Girls Hostel is managed under Hostel. Want phone number? (Yes/No)",
    "Who is the Boys Hostel warden?": "Boys Hostel is managed under Hostel. Want phone number? (Yes/No)",
    "How to reach Auditorium from Main Gate?": "Use Pathfinding tab with UCS or A* to get shortest path.",
    "Distance from Cafeteria to Admin Block?": f"{distances['Cafeteria']['Admin Block']} meters",
    "Nearest gate to Admin Block?": "Security Entrance is the nearest to Admin Block.",
    "Food Court meal timings?": location_timings.get("Food Court", {}).get("breakfast", {}).get("start", "?") + " - " + location_timings.get("Food Court", {}).get("dinner", {}).get("end", "?"),
    "Is Mini Mart open now?": "Mini Mart timings are: " + location_timings.get("Mini Mart", {}).get("start", "?") + " to " + location_timings.get("Mini Mart", {}).get("end", "?")
}

# --- Utility Functions ---
```

```
def get_location_timing(location):
    timings = location_timings.get(location)
    if not timings:
        return "No timing information available"
    if "start" in timings and "end" in timings:
        return f"Open from {timings['start']} to {timings['end']}"
    if "breakfast" in timings:
        return (
            f"Breakfast: {timings['breakfast']['start']} - {timings['breakfast']['end']}\n"
            f"Lunch: {timings['lunch']['start']} - {timings['lunch']['end']}\n"
            f"Dinner: {timings['dinner']['start']} - {timings['dinner']['end']}"
        )
    return "No timing information available"

def get_contact_info_from_answer(answer):
    keywords = {
        "admission": "Admission",
        "registrar": "Registrar",
        "finance": "Finance",
        "librarian": "Library",
        "medical": "Medical Center",
        "it support": "IT Support",
        "sports": "Sports & P.E",
        "girls hostel": "Girls",
        "boys hostel": "Boys"
    }
    for k, v in keywords.items():
        if k in answer.lower():
            # Use fuzzy matching to find closest key in contacts
            available_keys = list(contacts.keys())
            closest = difflib.get_close_matches(v, available_keys, n=1)
            if closest:
                found_key = closest[0]
                info = contacts.get(found_key, {}).get(v)
                if info is None:
                    info = contacts.get(found_key)
                if isinstance(info, list):
                    return "\n".join(f"{p['name']} - {p['phone']}" for p in info)
                if isinstance(info, dict):
                    return f"{info['name']} - {info['phone']}"
            return "Contact info not found."
    return "No contact info available for this query."

# Search algorithms

def bfs(graph, start, goal):
    visited = set()
    queue = deque([[start]])
    while queue:
        path = queue.popleft()
        node = path[-1]
        if node == goal:
            return path, len(path)-1
        if node not in visited:
            visited.add(node)
            for neighbor in graph.get(node, {}):
                if neighbor not in visited:
                    queue.append(path + [neighbor])
    return None, None

def dfs(graph, start, goal):
    visited = set()
    stack = [[start]]
    while stack:
        path = stack.pop()
        node = path[-1]
        if node == goal:
            return path, len(path)-1
        if node not in visited:
            visited.add(node)
            for neighbor in graph.get(node, {}):
                if neighbor not in visited:
                    stack.append(path + [neighbor])
    return None, None

def ucs(graph, start, goal):
    visited = set()
    pq = PriorityQueue()
    pq.put((0, [start]))
    while not pq.empty():
        cost, path = pq.get()
        node = path[-1]
        if node == goal:
            return path, cost
        if node not in visited:
            visited.add(node)
            for neighbor, weight in graph.get(node, {}).items():
                if neighbor not in visited:
                    pq.put((cost + weight, path + [neighbor]))
    return None, None

def a_star(graph, heuristics, start, goal):
    frontier = PriorityQueue()
    frontier.put((0, start))
    came_from = {start: None}
    cost_so_far = {start: 0}
    while not frontier.empty():
        _, current = frontier.get()
        if current == goal:
            break
        for neighbor, dist in graph.get(current, {}).items():
            new_cost = cost_so_far[current] + dist
            if neighbor not in cost_so_far or new_cost < cost_so_far[neighbor]:
                cost_so_far[neighbor] = new_cost
                priority = new_cost + heuristics.get(neighbor, 0)
                frontier.put((priority, neighbor))
                came_from[neighbor] = current
    if goal not in came_from and goal != start:
        return None, None
    node = goal
    path = []
    while node is not None:
        path.append(node)
        node = came_from.get(node)
    path.reverse()
    return path, cost_so_far.get(goal, 0)

def build_heuristics(goal):
    heur = {}
    if goal not in NODE_COORDS:
        return heur
    gx, gy = NODE_COORDS[goal]
    for name, (x, y) in NODE_COORDS.items():
        heur[name] = math.hypot(x-gx, y-gy)
    return heur

def compute_display_distance(path):
    if not path or len(path)<2:
        return 0
    total = 0
    for i in range(len(path)-1):
        total += distances[path[i]][path[i+1]]
    return total

def draw_path_on_map(path):
    global path_lines
    for line in path_lines:
        canvas.delete(line)
    path_lines.clear()
    if not path or len(path)<2:
        return
    for i in range(len(path)-1):
        x1, y1 = NODE_COORDS[path[i]]
        x2, y2 = NODE_COORDS[path[i+1]]
        line = canvas.create_line(x1, y1, x2, y2, fill="blue", width=4)
        path_lines.append(line)
    for node in path:
        x, y = NODE_COORDS[node]
        circ = canvas.create_oval(x-8, y-8, x+8, y+8, fill="blue", outline="")
        path_lines.append(circ)
    # Highlight start and end
    sx, sy = NODE_COORDS[path[0]]
    ex, ey = NODE_COORDS[path[-1]]
    path_lines.append(canvas.create_oval(sx-12, sy-12, sx+12, sy+12, outline="red", width=3))
    path_lines.append(canvas.create_oval(ex-12, ey-12, ex+12, ey+12, outline="green", width=3))

# Main Tkinter Application

root = tk.Tk()
root.title("Campus Digital Path Assistant")
```

```
panes = ttk.PanedWindow(root, orient="horizontal")

panes.pack(fill="both", expand=True)


left_panel = ttk.Frame(panes)

panes.add(left_panel, weight=1)


welcome_label = ttk.Label(left_panel, text="Welcome to Chanakya's Digital Path", font=("Arial",
16))

welcome_label.pack(pady=10)


notebook = ttk.Notebook(left_panel)

notebook.pack(fill="both", expand=True)


# Q&A Tab with Dropdown Question Selection


qa_tab = ttk.Frame(notebook)

notebook.add(qa_tab, text="Q&A")


ttk.Label(qa_tab, text="Select a question:", font=("Arial", 12)).pack(pady=6)


question_var = tk.StringVar()

qa_questions = list(qa_pairs.keys())

question_cb = ttk.Combobox(qa_tab, textvariable=question_var, values=qa_questions,
state="readonly", width=65)

question_cb.pack(padx=10, pady=6)


answer_label = ttk.Label(qa_tab, text="", font=("Arial", 11), wraplength=450, justify="left")

answer_label.pack(padx=10, pady=8)


def on_qa_ask():

    q = question_var.get()

    if not q:
        answer_label.config(text="Please select a question.")
        return
    answer = qa_pairs.get(q, "No answer available.")
    answer_label.config(text=answer)

    if '(yes/no)' in answer.lower():
        wants_phone = mb.askyesno("Question", "Do you want the phone number?")
        if wants_phone:
            contact_info = get_contact_info_from_answer(answer)
            answer_label.config(text=answer + "\n\nContact info:\n" + contact_info)
        else:
            answer_label.config(text=answer + "\n\nYou chose not to see contact info.")

ask_btn = ttk.Button(qa_tab, text="Get Answer", command=on_qa_ask)
ask_btn.pack(pady=5)

# Staff Contacts Tab

staff_tab = ttk.Frame(notebook)
notebook.add(staff_tab, text="Staff Contacts")

ttk.Label(staff_tab, text="Select Department:", font=("Arial", 12)).pack(pady=8)

staff_main_var = tk.StringVar()
staff_main_cb = ttk.Combobox(staff_tab, values=list(contacts.keys()), textvariable=staff_main_var, state="readonly", width=40)
staff_main_cb.pack(pady=6)

ttk.Label(staff_tab, text="Select Subcategory:", font=("Arial", 12)).pack(pady=8)

staff_sub_var = tk.StringVar()
staff_sub_cb = ttk.Combobox(staff_tab, values=[], textvariable=staff_sub_var, state="disabled", width=40)
```

```
staff_sub_cb.pack(pady=6)

staff_output = tk.Text(staff_tab, width=60, height=12, state="disabled", wrap="word")
staff_output.pack(pady=10)

def on_staff_main_select(event=None):
    dept = staff_main_var.get()
    staff_output.config(state="normal")
    staff_output.delete("1.0", tk.END)
    if dept:
        info = contacts.get(dept)
        if isinstance(info, dict) and "name" in info and "phone" in info:
            staff_sub_cb.config(state="disabled", values=[])
            staff_sub_var.set('')
            staff_output.insert(tk.END, f"{info['name']} - {info['phone']}")
        elif isinstance(info, dict):
            subs = list(info.keys())
            staff_sub_cb.config(state="readonly", values=subs)
            staff_sub_cb.set('')
            staff_output.insert(tk.END, f"Select a subcategory for {dept}")
        else:
            staff_sub_cb.config(state="disabled", values=[])
            staff_sub_var.set('')
            staff_output.insert(tk.END, "No contact info available.")
    else:
        staff_sub_cb.config(state="disabled", values=[])
        staff_sub_var.set('')
    staff_output.config(state="disabled")

def on_staff_sub_select(event=None):
    dept = staff_main_var.get()
    sub = staff_sub_var.get()
    staff_output.config(state="normal")
    staff_output.delete("1.0", tk.END)
    if dept and sub:
        info = contacts.get(dept, {}).get(sub)
        if isinstance(info, list):
            for p in info:
                staff_output.insert(tk.END, f"{p['name']} - {p['phone']}\n")
        elif isinstance(info, dict):
            staff_output.insert(tk.END, f"{info['name']} - {info['phone']}")
        else:
            staff_output.insert(tk.END, "No contact info available.")
    else:
        staff_output.insert(tk.END, "Please select both department and subcategory.")
    staff_output.config(state="disabled")

staff_main_cb.bind("<<ComboboxSelected>>", on_staff_main_select)
staff_sub_cb.bind("<<ComboboxSelected>>", on_staff_sub_select)

# Timings Tab

timings_tab = ttk.Frame(notebook)
notebook.add(timings_tab, text="Timings")

ttk.Label(timings_tab, text="Select Location:", font=("Arial", 12)).pack(pady=(20, 5))
timings_location_var = tk.StringVar()
timings_location_cb = ttk.Combobox(timings_tab, values=list(location_timings.keys()),
state="readonly", textvariable=timings_location_var, width=40)
timings_location_cb.pack(pady=(0, 10))

timings_output_label = ttk.Label(timings_tab, text="", font=("Arial", 11), foreground="blue",
justify="left")
timings_output_label.pack(pady=15)
```

```
def show_timings():
    loc = timings_location_var.get().strip()
    if not loc:
        timings_output_label.config(text="Please select a location.")
        return
    result = get_location_timing(loc)
    timings_output_label.config(text=result)

timings_get_btn = ttk.Button(timings_tab, text="Get Timings", command=show_timings)
timings_get_btn.pack()

# Pathfinding Tab

path_tab = ttk.Frame(notebook)
notebook.add(path_tab, text="Pathfinding")

ttk.Label(path_tab, text="Start Location:").grid(row=0, column=0, sticky="w", padx=5, pady=5)
ttk.Label(path_tab, text="End Location:").grid(row=1, column=0, sticky="w", padx=5, pady=5)
ttk.Label(path_tab, text="Algorithm:").grid(row=2, column=0, sticky="w", padx=5, pady=5)

start_choice = tk.StringVar()
start_cb = ttk.Combobox(path_tab, textvariable=start_choice, values=list(distances.keys()),
state="readonly", width=40)
start_cb.grid(row=0, column=1, padx=5, pady=5)

end_choice = tk.StringVar()
end_cb = ttk.Combobox(path_tab, textvariable=end_choice, values=list(distances.keys()),
state="readonly", width=40)
end_cb.grid(row=1, column=1, padx=5, pady=5)

algo_choice = tk.StringVar()
algo_cb = ttk.Combobox(path_tab, textvariable=algo_choice, values=["Auto", "BFS", "DFS", "UCS",
"A*"], state="readonly", width=40)
```

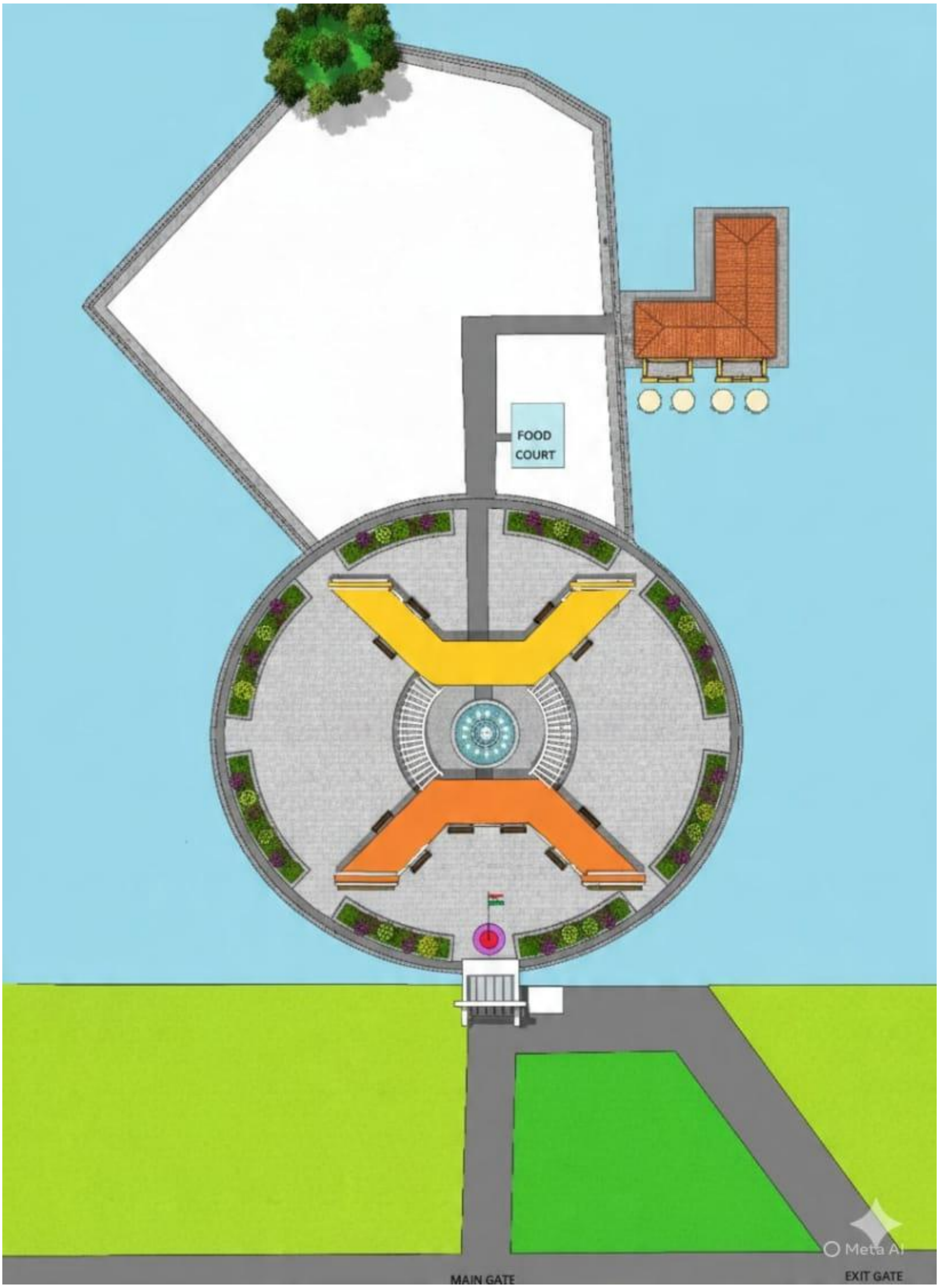
```
algo_cb.set("Auto")
algo_cb.grid(row=2, column=1, padx=5, pady=5)

find_path_btn = ttk.Button(path_tab, text="Find Path", command=lambda: run_algorithm())
find_path_btn.grid(row=3, column=0, columnspan=2, pady=10)

result_text = tk.Text(path_tab, width=60, height=10, wrap="word")
result_text.grid(row=4, column=0, columnspan=2, padx=10, pady=10)

def run_algorithm():
    start = start_choice.get().strip()
    end = end_choice.get().strip()
    algo = algo_choice.get().strip()
    result_text.delete("1.0", tk.END)
    if start not in distances or end not in distances:
        result_text.insert(tk.END, "Invalid start or end location.\n")
        return
    heuristics = build_heuristics(end)
    chosen = algo if algo != "Auto" else ("A*" if heuristics else "UCS")
    if chosen == "BFS":
        path, cost = bfs(distances, start, end)
    elif chosen == "DFS":
        path, cost = dfs(distances, start, end)
    elif chosen == "UCS":
        path, cost = ucs(distances, start, end)
    elif chosen == "A*":
        path, cost = a_star(distances, heuristics, start, end)
    else:
        path, cost = None, None
    if path:
        display_dist = compute_display_distance(path) if chosen in ("BFS", "DFS") else cost
        time_minutes = round(display_dist / 83, 1)  # ~5 km/h walking speed
        result_text.insert(tk.END, f"Algorithm: {chosen}\nDistance: {display_dist} meters\nEstimated time: {time_minutes} min\nPath: {' → '.join(path)}\n")
        draw_path_on_map(path)
    else:
        result_text.insert(tk.END, "No path found.\n")

# Canvas with map

canvas_frame = ttk.Frame(panes)
panes.add(canvas_frame, weight=2)

canvas = tk.Canvas(canvas_frame, width=800, height=800, bg="white", scrollregion=(0, 0, 1200, 1200))
hbar = ttk.Scrollbar(canvas_frame, orient="horizontal", command=canvas.xview)
vbar = ttk.Scrollbar(canvas_frame, orient="vertical", command=canvas.yview)
canvas.config(xscrollcommand=hbar.set, yscrollcommand=vbar.set)
canvas.grid(row=0, column=0, sticky="nsew")
hbar.grid(row=1, column=0, sticky="ew")
vbar.grid(row=0, column=1, sticky="ns")
canvas_frame.rowconfigure(0, weight=1)
canvas_frame.columnconfigure(0, weight=1)

# Load map image - ensure 'chanakya_map.jpg' is in your working directory

try:
    map_img = Image.open(MAP_PATH)
    map_tk = ImageTk.PhotoImage(map_img)
    canvas_map = canvas.create_image(0, 0, anchor="nw", image=map_tk)
except Exception as e:
    mb.showerror("Image Load Error", f"Failed to load map image: {e}")

# Draw nodes on canvas
```

```
for name, (x, y) in NODE_COORDS.items():
    canvas.create_oval(x - 7, y - 7, x + 7, y + 7, outline="black", fill="lightgray", width=2)
    canvas.create_text(x, y - 10, text=name, font=("Arial", 9), fill="black")

path_lines = []
selected_nodes = []

def on_map_click(event):
    x, y = event.x, event.y
    nearest = None
    min_dist = float('inf')
    for key, (nx, ny) in NODE_COORDS.items():
        dist = math.hypot(x - nx, y - ny)
        if dist < min_dist and dist < 30:
            min_dist = dist
            nearest = key
    if nearest:
        selected_nodes.append(nearest)
        if len(selected_nodes) > 2:
            selected_nodes.pop(0)
        if selected_nodes:
            start_choice.set(selected_nodes[0])
        if len(selected_nodes) > 1:
            end_choice.set(selected_nodes[1])

canvas.bind("<Button-1>", on_map_click)

root.mainloop()
```

OUTPUT DEMO VEDIO

Screen Recording 2025-09-21 204241.m

MAP USED IN CODE


FOOD COURT
Meta AI
MAIN GATE
EXIT GATE