A logo of a university

Description automatically generated**COMSATS UNIVERSITY ISLAMABAD ATTOCK CAMPUS**

**Artificial intelligence**

**Project code and output**

**Name and Reg No:**

Muhammad Fassi Ur Rehman (FA23 -BAI- 042)

Absar Ahmed (FA23 -BAI- 038)

Muhammad Taha (FA23 -BAI- 012)

**Submitted to:** Sir Babar Ali

**Date: June 2, 2025**

**Code:**

import tkinter as tk

from tkinter import messagebox

from typing import  Tuple

class Node:

    def \_\_init\_\_(self, state: Tuple[int, int], parent, actions, totalcost: int, heuristic: int):

        self.state = state

        self.parent = parent

        self.actions = actions

        self.totalcost = totalcost

        self.heuristic = heuristic

def manhattan\_distance(a: Tuple[int, int], b: Tuple[int, int]) -> int:

    return abs(a[0] - b[0]) + abs(a[1] - b[1])

def A\_star(start, goal, grid):

    rows = len(grid)

    cols = len(grid[0])

    graph = {}

    # Build graph with valid nodes

    for x in range(rows):

        for y in range(cols):

            if grid[x][y] == '#':

                continue

            actions = []

            for direction in [(-1, 0), (1, 0), (0, -1), (0, 1)]:

                dx = direction[0]

                dy = direction[1]

                nx = x + dx

                ny = y + dy

                if 0 <= nx < rows and 0 <= ny < cols and grid[nx][ny] != '#':

                    actions.append(((nx, ny), 1))

            node = Node(

                state=(x, y),

                parent=None,

                actions=actions,

                totalcost=float('inf'),

                heuristic=manhattan\_distance((x, y), goal)

            )

            graph[(x, y)] = node

    if start not in graph or goal not in graph:

        return []

    graph[start].totalcost = 0

    frontier = [(start, 0)]

    explored = []

    while frontier:

        # Find node with lowest cost + heuristic

        best\_index = 0

        best\_node = frontier[0]

        for i in range(1, len(frontier)):

            pos, cost = frontier[i]

            score = cost + graph[pos].heuristic

            best\_score = best\_node[1] + graph[best\_node[0]].heuristic

            if score < best\_score:

                best\_node = frontier[i]

                best\_index = i

        current\_node, current\_cost = frontier.pop(best\_index)

        explored.append(current\_node)

        if current\_node == goal:

            path = [goal]

            while graph[path[-1]].parent is not None:

                path.append(graph[path[-1]].parent)

            return path[::-1]

        for child, cost in graph[current\_node].actions:

            new\_cost = current\_cost + cost

            if child not in explored and new\_cost < graph[child].totalcost:

                graph[child].totalcost = new\_cost

                graph[child].parent = current\_node

                frontier.append((child, new\_cost))

    return []

class SearchAndRescueGUI:

    def \_\_init\_\_(self, master, grid, start, hospital, victims\_info):

        self.master = master

        self.master.title("Search and Rescue Simulation")

        self.grid = grid

        self.start = self.current\_position = start

        self.hospital = hospital

        self.victims\_info = victims\_info.copy()

        self.treated\_victims = set()

        self.cell\_size = 100

        # Color mapping for different criticality levels

        self.criticality\_colors = {

            1: "#FFCC00",  # Light yellow for low criticality

            2: "#FF9900",  # Orange

            3: "#FF6600",  # Dark orange

            4: "#FF3300",  # Red-orange

            5: "#FF0000"   # Bright red for highest criticality

        }

        self.canvas = tk.Canvas(master, width=len(grid[0])\*self.cell\_size, height=len(grid)\*self.cell\_size)

        self.canvas.pack()

        self.controls = tk.Frame(master)

        self.controls.pack()

        tk.Button(self.controls, text="Start Simulation", command=self.start\_simulation).pack(side=tk.LEFT)

        tk.Button(self.controls, text="Next Step", command=self.next\_step).pack(side=tk.LEFT)

        tk.Button(self.controls, text="Reset", command=self.reset\_simulation).pack(side=tk.LEFT)

        self.status\_label = tk.Label(master, text="Ready to start simulation")

        self.status\_label.pack()

        self.draw\_grid()

    def draw\_grid(self):

        self.canvas.delete("all")

        for x in range(len(self.grid)):

            for y in range(len(self.grid[0])):

                x1, y1 = y \* self.cell\_size, x \* self.cell\_size

                color = "white"

                if (x, y) == self.start:

                    color = "lightblue"

                elif (x, y) == self.hospital:

                    color = "pink"

                elif self.grid[x][y] == '#':

                    color = "gray"

                elif (x, y) in self.victims\_info and (x, y) not in self.treated\_victims:

                    crit = self.victims\_info[(x, y)]['criticality']

                    color = self.criticality\_colors.get(min(max(crit, 1), 5), "#FF0000")  # Default to red if out of range

                self.canvas.create\_rectangle(x1, y1, x1 + self.cell\_size, y1 + self.cell\_size,

                                           fill=color, outline="black")

                # Add text labels

                if (x, y) == self.start:

                    self.canvas.create\_text(x1 + self.cell\_size/2, y1 + self.cell\_size/2, text="R",font=("Arial", 16))

                elif (x, y) == self.hospital:

                    self.canvas.create\_text(x1 + self.cell\_size/2, y1 + self.cell\_size/2, text="H",font=("Arial", 16))

                elif self.grid[x][y] == '#':

                    self.canvas.create\_text(x1 + self.cell\_size/2, y1 + self.cell\_size/2, text="#",font=("Arial", 16))

                elif (x, y) in self.victims\_info and (x, y) not in self.treated\_victims:

                    self.canvas.create\_text(x1 + self.cell\_size/2, y1 + self.cell\_size/2,

                                          text=self.victims\_info[(x, y)]['label'], fill="white",font=("Arial", 16))

        # Draw robot position

        if self.current\_position:

            x, y = self.current\_position[1]\*self.cell\_size + self.cell\_size/2, self.current\_position[0]\*self.cell\_size + self.cell\_size/2

            self.canvas.create\_oval(x-10, y-10, x+10, y+10, fill="blue")

    def start\_simulation(self):

        self.current\_position = self.start

        self.treated\_victims = set()

        self.victims\_info = victims\_info.copy()

        self.full\_path, self.movement\_events = self.search\_and\_rescue(self.grid.copy(), self.start, self.hospital, self.victims\_info.copy())

        self.step\_index = 0

        self.status\_label.config(text="Simulation started - press Next Step")

        self.draw\_grid()

    def next\_step(self):

        if not hasattr(self, 'full\_path') or self.step\_index >= len(self.full\_path):

            self.status\_label.config(text="Simulation complete!")

            messagebox.showinfo("Done", "All victims rescued and robot returned!")

            return

        self.current\_position = self.full\_path[self.step\_index]

        action\_message = self.movement\_events[self.step\_index]

        if "Treating" in action\_message or "Picked up" in action\_message:

            for pos, info in self.victims\_info.items():

                if info['label'] in action\_message:

                    self.treated\_victims.add(pos)

                    break

        self.status\_label.config(text=action\_message)

        self.step\_index += 1

        self.draw\_grid()

    def reset\_simulation(self):

        self.current\_position = self.start

        self.treated\_victims = set()

        self.step\_index = 0

        if hasattr(self, 'full\_path'):

            del self.full\_path

        self.status\_label.config(text="Simulation reset")

        self.draw\_grid()

    def search\_and\_rescue(self, grid, start, hospital, victims\_info):

        path\_log = [start]

        event\_log = ["Starting at base"]

        current\_position = start

        while len(victims\_info) > 0:

            # Step 1: Choose highest criticality victim, break ties by distance

            best\_victim = None

            best\_criticality = -1

            best\_distance = float('inf')

            for pos in victims\_info:

                crit = victims\_info[pos]['criticality']

                dist = manhattan\_distance(current\_position, pos)

                if crit > best\_criticality:

                    best\_victim = pos

                    best\_criticality = crit

                    best\_distance = dist

                elif crit == best\_criticality and dist < best\_distance:

                    best\_victim = pos

                    best\_distance = dist

            label = victims\_info[best\_victim]['label']

            # Step 2: Move to victim

            path\_to\_victim = A\_star(current\_position, best\_victim, grid)

            for i in range(1, len(path\_to\_victim)):

                path\_log.append(path\_to\_victim[i])

                event\_log.append("Moving to " + label)

            current\_position = best\_victim

            # Step 3: Treat or Pick Up

            if victims\_info[best\_victim]['treat\_on\_spot']:

                event\_log[-1] = "Treating " + label + " (criticality: " + str(best\_criticality) + ") on spot"

            else:

                event\_log[-1] = "Picked up " + label + " (criticality: " + str(best\_criticality) + ")"

            # Step 4: Deliver to hospital

                path\_to\_hospital = A\_star(current\_position, hospital, grid)

                for i in range(1, len(path\_to\_hospital)):

                    path\_log.append(path\_to\_hospital[i])

                    event\_log.append("Delivering " + label + " to hospital")

                current\_position = hospital

                event\_log[-1] = "Delivered " + label

        # Step 5: Remove treated victim

            del victims\_info[best\_victim]

    # Step 6: Return to base

        path\_home = A\_star(current\_position, start, grid)

        for i in range(1, len(path\_home)):

            path\_log.append(path\_home[i])

            event\_log.append("Returning to base")

        return path\_log, event\_log

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

    grid = [

        ['R', '\_', '\_', '\_', 'V1'],

        ['#', '#', '\_', '#', '\_'],

        ['\_', 'v5', '\_', 'V4', 'V2'],

        ['\_', '#', '\_', '#', '\_'],

        ['\_', 'V3', '\_', '\_', 'H']

    ]

    start = (0, 0)

    hospital = (4, 4)

    victims\_info = {

        (0, 4): {'criticality': 2, 'treat\_on\_spot': True,  'label': 'V1'},

        (2, 4): {'criticality': 5, 'treat\_on\_spot': False, 'label': 'V2'},

        (4, 1): {'criticality': 3, 'treat\_on\_spot': False, 'label': 'V3'},

        (2, 1): {'criticality': 9, 'treat\_on\_spot': False, 'label': 'V5'},

        (2, 3): {'criticality': 6, 'treat\_on\_spot': True, 'label': 'V4'},

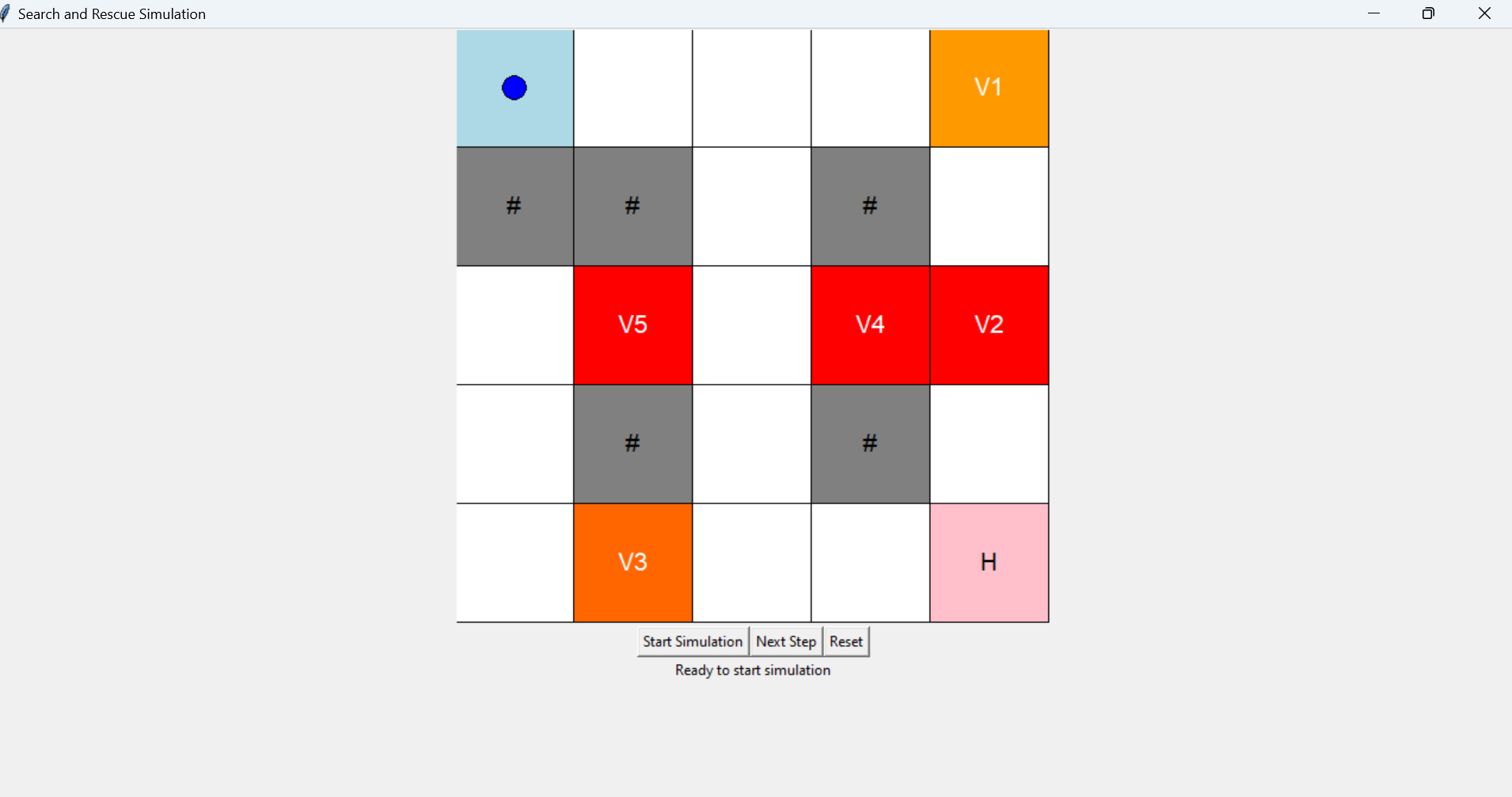
    }

    root = tk.Tk()

    SearchAndRescueGUI(root, grid, start, hospital, victims\_info)

    root.mainloop()

**output:**

A screenshot of a computer game

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer game

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer game

AI-generated content may be incorrect.A screenshot of a game

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.