Delivery Simulator

With city presets: Bhopal, Indore, Shivpuri, Jabalpur

"""

from collections import deque

import heapq

import random

import time

# -------------------------

# City Map Class

# -------------------------

class CityMap:

def \_\_init\_\_(self, width, height, name="City"):

self.width = width

self.height = height

self.name = name

self.walls = set()

self.terrain\_cost = {}

self.traffic = {}

self.deliveries = []

def add\_wall(self, x, y):

self.walls.add((x, y))

def set\_terrain(self, x, y, cost):

self.terrain\_cost[(x, y)] = cost

def add\_traffic(self, t, x, y):

self.traffic.setdefault(t, set()).add((x, y))

def add\_delivery(self, start\_x, start\_y, end\_x, end\_y):

self.deliveries.append(((start\_x, start\_y), (end\_x, end\_y)))

def is\_open(self, x, y, t=0):

if x < 0 or x >= self.width or y < 0 or y >= self.height:

return False

if (x, y) in self.walls:

return False

if t in self.traffic and (x, y) in self.traffic[t]:

return False

return True

def get\_cost(self, x, y):

return self.terrain\_cost.get((x, y), 1)

def show(self, agent\_pos=None, route=None, clock=0):

print("\n" + "=" \* (self.width \* 3 + 2))

print(f"{self.name.upper()} MAP (Time: {clock})")

print("=" \* (self.width \* 3 + 2))

for y in range(self.height):

row = "|"

for x in range(self.width):

spot = (x, y)

if agent\_pos and spot == agent\_pos:

row += " R "

elif spot in self.walls:

row += " # "

elif clock in self.traffic and spot in self.traffic[clock]:

row += " T "

elif route and spot in route:

row += " · "

elif spot in self.terrain\_cost and self.terrain\_cost[spot] > 1:

row += f" {self.terrain\_cost[spot]} "

else:

row += " . "

row += "|"

print(row)

print("=" \* (self.width \* 3 + 2))

print("Legend: R=Robot #=Wall T=Traffic ·=Route Number=Cost\n")

# -------------------------

# Search Algorithms

# -------------------------

def bfs(city, start, goal):

q = deque([(start[0], start[1], 0, [start])])

visited = set([start])

explored = 0

while q:

x, y, c, path = q.popleft()

explored += 1

if (x, y) == goal:

return path, c, explored

for dx, dy in [(0, 1), (1, 0), (0, -1), (-1, 0)]:

nx, ny = x + dx, y + dy

if city.is\_open(nx, ny) and (nx, ny) not in visited:

visited.add((nx, ny))

q.append((nx, ny, c + city.get\_cost(nx, ny), path + [(nx, ny)]))

return [], float('inf'), explored

def ucs(city, start, goal):

pq = [(0, start[0], start[1], [start])]

visited = set()

explored = 0

while pq:

cost, x, y, path = heapq.heappop(pq)

explored += 1

if (x, y) == goal:

return path, cost, explored

if (x, y) in visited:

continue

visited.add((x, y))

for dx, dy in [(0, 1), (1, 0), (0, -1), (-1, 0)]:

nx, ny = x + dx, y + dy

if city.is\_open(nx, ny) and (nx, ny) not in visited:

heapq.heappush(pq, (cost + city.get\_cost(nx, ny), nx, ny, path + [(nx, ny)]))

return [], float('inf'), explored

def astar(city, start, goal):

def h(x1, y1, x2, y2):

return abs(x1 - x2) + abs(y1 - y2)

pq = [(h(\*start, \*goal), 0, start[0], start[1], [start])]

visited = set()

explored = 0

while pq:

\_, cost, x, y, path = heapq.heappop(pq)

explored += 1

if (x, y) == goal:

return path, cost, explored

if (x, y) in visited:

continue

visited.add((x, y))

for dx, dy in [(0, 1), (1, 0), (0, -1), (-1, 0)]:

nx, ny = x + dx, y + dy

if city.is\_open(nx, ny) and (nx, ny) not in visited:

new\_c = cost + city.get\_cost(nx, ny)

heapq.heappush(pq, (new\_c + h(nx, ny, goal[0], goal[1]), new\_c, nx, ny, path + [(nx, ny)]))

return [], float('inf'), explored

def random\_local(city, start, goal, tries=4):

best\_route, best\_cost = [], float('inf')

explored\_total = 0

for t in range(tries):

x, y = start

path = [start]

cost, explored = 0, 0

while (x, y) != goal and explored < 100:

explored += 1

options = []

for dx, dy in [(0, 1), (1, 0), (0, -1), (-1, 0)]:

nx, ny = x + dx, y + dy

if city.is\_open(nx, ny):

step\_cost = city.get\_cost(nx, ny)

dist = abs(nx - goal[0]) + abs(ny - goal[1])

options.append((step\_cost + dist, step\_cost, nx, ny))

if not options:

break

if random.random() < 0.8:

\_, step\_cost, nx, ny = min(options)

else:

\_, step\_cost, nx, ny = random.choice(options)

path.append((nx, ny))

cost += step\_cost

x, y = nx, ny

if (x, y) == goal and cost < best\_cost:

best\_route, best\_cost = path, cost

explored\_total += explored

return best\_route, best\_cost, explored\_total

# -------------------------

# Courier

# -------------------------

class Courier:

def \_\_init\_\_(self, city):

self.city = city

self.x, self.y = 0, 0

self.clock = 0

self.fuel = 100

self.history = []

self.done = 0

def status(self):

print(f"Location: ({self.x}, {self.y}) Time: {self.clock} Fuel: {self.fuel} Delivered: {self.done}\n")

def run\_delivery(self, start, end, algo, animate=True):

self.x, self.y = start

print(f"\nStarting delivery {start} -> {end} in {self.city.name} using {algo.upper()}")

if algo == "bfs":

route, plan\_cost, explored = bfs(self.city, start, end)

elif algo == "ucs":

route, plan\_cost, explored = ucs(self.city, start, end)

elif algo == "astar":

route, plan\_cost, explored = astar(self.city, start, end)

else:

route, plan\_cost, explored = random\_local(self.city, start, end)

if not route:

msg = f"No route from {start} to {end}"

self.history.append("Sorry! " + msg)

print(msg)

return 0, explored, False

if animate:

print("Planned route length:", len(route), "cost estimate:", plan\_cost)

time.sleep(0.6)

total\_cost = 0

for step\_idx, (nx, ny) in enumerate(route[1:], 1):

now = self.clock + step\_idx

if animate:

self.city.show((self.x, self.y), route, now)

self.status()

time.sleep(0.3)

if not self.city.is\_open(nx, ny, now):

self.history.append(f" Blocked at {nx},{ny} at t={now}")

new\_r, new\_c, new\_e = random\_local(self.city, (self.x, self.y), end, tries=6)

explored += new\_e

if new\_r:

route = [(self.x, self.y)] + new\_r

self.history.append("New route found")

continue

else:

self.history.append("Stuck, delivery failed")

break

move\_cost = self.city.get\_cost(nx, ny)

self.x, self.y = nx, ny

total\_cost += move\_cost

self.fuel -= move\_cost

self.clock += max(0, len(route) - 1)

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

self.history.append(f"Delivered in {self.city.name}! Cost={total\_cost}")

self.done += 1

return total\_cost, explored, True

else:

return total\_cost, explored, False

# -------------------------

# City Presets

# -------------------------

def bhopal\_city():

city = CityMap(8, 6, "Bhopal")

for w in [(1, 1), (2, 1), (3, 1), (4, 4)]:

city.add\_wall(\*w)

city.set\_terrain(5, 2, 3)

for t in range(3, 6):

city.add\_traffic(t, 2, 0)

city.add\_delivery(0, 0, 7, 5)

return city

def indore\_city():

city = CityMap(7, 7, "Indore")

for w in [(2, 2), (2, 3), (2, 4), (5, 5)]:

city.add\_wall(\*w)

city.set\_terrain(4, 1, 4)

for t in range(4, 9):

city.add\_traffic(t, 3, 3)

city.add\_delivery(0, 0, 6, 6)

return city

def shivpuri\_city():

city = CityMap(6, 8, "Shivpuri")

for w in [(1, 6), (2, 6), (3, 6), (4, 6)]:

city.add\_wall(\*w)

city.set\_terrain(3, 2, 2)

for t in range(2, 5):

city.add\_traffic(t, 4, 0)

city.add\_delivery(0, 0, 5, 7)

return city

def jabalpur\_city():

city = CityMap(9, 5, "Jabalpur")

for w in [(7, 1), (7, 2), (7, 3)]:

city.add\_wall(\*w)

city.set\_terrain(2, 2, 5)

for t in range(6, 10):

city.add\_traffic(t, 5, 4)

city.add\_delivery(0, 0, 8, 4)

return city

# -------------------------

# Menu / CLI

# -------------------------

def menu():

city\_options = {

"1": bhopal\_city,

"2": indore\_city,

"3": shivpuri\_city,

"4": jabalpur\_city,

}

print("DELIVERY SIMULATOR")

print("=" \* 40)

while True:

print("\nSelect City:")

print("1. Bhopal")

print("2. Indore")

print("3. Shivpuri")

print("4. Jabalpur")

print("5. Quit")

choice = input("Choose (1-5): ").strip()

if choice == "5":

print("Bye from Delivery Simulator!")

break

if choice not in city\_options:

print("Invalid selection.")

continue

current\_city = city\_options[choice]()

bot = Courier(current\_city)

while True:

print(f"\n You are in {current\_city.name}!")

print("1. Show city map")

print("2. Run one delivery")

print("3. Compare algorithms")

print("4. Show courier status")

print("5. Show log (last 10)")

print("6. Back to city menu")

sub = input("Choose (1-6): ").strip()

if sub == "1":

current\_city.show()

input("Press Enter...")

elif sub == "2":

start, end = current\_city.deliveries[0]

print("Algorithms: 1=BFS 2=UCS 3=A\* 4=Local")

alg = input("Pick (1-4): ").strip()

mapping = {"1": "bfs", "2": "ucs", "3": "astar", "4": "local"}

algo = mapping.get(alg, "local")

animate = input("Show animation? (y/n): ").strip().lower().startswith("y")

cost, nodes, ok = bot.run\_delivery(start, end, algo, animate)

print(f"Result: Cost={cost}, Explored={nodes}, Success={ok}")

input("Press Enter...")

elif sub == "3":

start, end = current\_city.deliveries[0]

for m in ["bfs", "ucs", "astar", "local"]:

bot.clock, bot.fuel = 0, 100

cost, nodes, ok = bot.run\_delivery(start, end, m, animate=False)

print(f"{m.upper():<6} | Cost={cost:<4} | Nodes={nodes:<4} | Success={ok}")

input("Press Enter...")

elif sub == "4":

bot.status()

input("Press Enter...")

elif sub == "5":

for log in bot.history[-10:]:

print(log)

input("Press Enter...")

elif sub == "6":

break

else:

print("Invalid.")

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

random.seed(1)

menu()