import time

import heapq

import random

from IPython.display import clear\_output

# Constants

FREE, OBSTACLE = 0, 1

ROBOT1, ROBOT2 = 4, 5

PICKUP1, DROP1 = 6, 7

PICKUP2, DROP2 = 8, 9

# Emojis for display

EMOJI\_MAP = {

FREE: "⬜",

OBSTACLE: "❌",

ROBOT1: "🤖",

ROBOT2: "👾",

PICKUP1: "🟩",

DROP1: "🟥",

PICKUP2: "🔵",

DROP2: "🟨",

}

# Generate a larger warehouse

def generate\_warehouse(rows=15, cols=15, obstacle\_ratio=0.2):

grid = [[FREE for \_ in range(cols)] for \_ in range(rows)]

# Place obstacles

obstacles = int(rows \* cols \* obstacle\_ratio)

while obstacles > 0:

r, c = random.randint(0, rows-1), random.randint(0, cols-1)

if grid[r][c] == FREE:

grid[r][c] = OBSTACLE

obstacles -= 1

# Place robot starts and goals

def place\_unique(vals):

while True:

r, c = random.randint(0, rows-1), random.randint(0, cols-1)

if grid[r][c] == FREE:

grid[r][c] = val

return (r, c)

robot1\_start = place\_unique(ROBOT1)

pickup1 = place\_unique(PICKUP1)

drop1 = place\_unique(DROP1)

robot2\_start = place\_unique(ROBOT2)

pickup2 = place\_unique(PICKUP2)

drop2 = place\_unique(DROP2)

return grid, robot1\_start, pickup1, drop1, robot2\_start, pickup2, drop2

# A\* pathfinding algorithm

def astar(grid, start, goal):

rows, cols = len(grid), len(grid[0])

heap = [(0, start)]

came\_from = {}

cost\_so\_far = {start: 0}

def heuristic(a, b):

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

while heap:

\_, current = heapq.heappop(heap)

if current == goal:

path = []

while current != start:

path.append(current)

current = came\_from[current]

return path[::-1]

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

ny, nx = current[0] + dy, current[1] + dx

neighbor = (ny, nx)

if 0 <= ny < rows and 0 <= nx < cols and grid[ny][nx] not in [OBSTACLE]:

new\_cost = cost\_so\_far[current] + 1

if neighbor not in cost\_so\_far or new\_cost < cost\_so\_far[neighbor]:

cost\_so\_far[neighbor] = new\_cost

priority = new\_cost + heuristic(neighbor, goal)

heapq.heappush(heap, (priority, neighbor))

came\_from[neighbor] = current

return None

# Print grid with dynamic updates

def print\_warehouse(grid, robot1\_pos=None, robot2\_pos=None):

clear\_output(wait=True)

for y in range(len(grid)):

row = ""

for x in range(len(grid[0])):

if (y, x) == robot1\_pos:

row += EMOJI\_MAP[ROBOT1]

elif (y, x) == robot2\_pos:

row += EMOJI\_MAP[ROBOT2]

else:

row += EMOJI\_MAP[grid[y][x]]

print(row)

print()

# Robot movement

def move\_robot(grid, path, carrying, robot\_id, pickup, drop):

robot\_emoji = EMOJI\_MAP[ROBOT1 if robot\_id == 1 else ROBOT2]

for pos in path:

print\_warehouse(grid, pos if robot\_id == 1 else None, pos if robot\_id == 2 else None)

y, x = pos

if (y, x) == pickup and not carrying:

carrying = True

print(f"{robot\_emoji} Robot {robot\_id} picked up the item!")

elif (y, x) == drop and carrying:

carrying = False

print(f"{robot\_emoji} Robot {robot\_id} dropped off the item!")

time.sleep(0.4)

return carrying

# Run the simulation

def run\_simulation():

grid, r1\_start, p1, d1, r2\_start, p2, d2 = generate\_warehouse()

print("🚀 Starting 2-Robot Warehouse Simulation...\n")

print\_warehouse(grid)

time.sleep(1)

# === Robot 1 ===

carrying = False

path1 = astar(grid, r1\_start, p1)

if path1:

carrying = move\_robot(grid, path1, carrying, 1, p1, d1)

r1\_curr = path1[-1]

else:

print("❌ Robot 1 cannot reach pickup.")

return

path2 = astar(grid, r1\_curr, d1)

if path2:

carrying = move\_robot(grid, path2, carrying, 1, p1, d1)

else:

print("❌ Robot 1 cannot reach drop point.")

return

print("✅ Robot 1 completed its task!")

time.sleep(2)

# === Robot 2 ===

carrying = False

path1 = astar(grid, r2\_start, p2)

if path1:

carrying = move\_robot(grid, path1, carrying, 2, p2, d2)

r2\_curr = path1[-1]

else:

print("❌ Robot 2 cannot reach pickup.")

return

path2 = astar(grid, r2\_curr, d2)

if path2:

carrying = move\_robot(grid, path2, carrying, 2, p2, d2)

else:

print("❌ Robot 2 cannot reach drop point.")

return

print("✅ Robot 2 completed its task!")

print("🏁 All tasks completed.")

# Run it!

run\_simulation()

**OUTPUT:**

❌❌⬜⬜⬜⬜⬜❌❌⬜⬜⬜⬜⬜⬜

❌❌⬜⬜❌❌⬜⬜❌⬜⬜⬜⬜⬜⬜

❌🔵⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜

⬜❌⬜⬜🟥❌⬜⬜⬜⬜⬜⬜❌🟩⬜

👾❌⬜⬜⬜❌⬜❌⬜⬜⬜❌⬜⬜⬜

⬜⬜⬜⬜⬜❌❌⬜⬜⬜⬜❌⬜⬜⬜

⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜❌⬜❌

⬜⬜⬜❌⬜⬜⬜❌⬜⬜⬜⬜⬜⬜⬜

❌⬜⬜⬜⬜⬜⬜⬜⬜⬜🤖⬜⬜⬜⬜

❌⬜⬜❌⬜⬜⬜⬜⬜⬜⬜⬜⬜❌⬜

⬜⬜⬜⬜⬜❌👾❌⬜⬜⬜⬜⬜❌❌

⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜❌⬜⬜⬜

⬜⬜⬜⬜⬜⬜⬜❌⬜❌⬜⬜⬜⬜⬜

❌⬜⬜⬜⬜⬜⬜❌⬜⬜⬜❌⬜⬜⬜

❌⬜⬜⬜⬜❌⬜❌⬜❌⬜❌❌❌⬜

👾 Robot 2 dropped off the item!

✅ Robot 2 completed its task!

🏁 All tasks completed.

