Fleet Manager Conflict Resolution Task

Task 2: Algorithm Design (Coding Exercise)

Problem:

The Fleet Management System (FMS) has sent the paths to robots without considering any scheduling constraints. The robots are moving along the path towards the conflicting region. Paths of robots are expressed as an array of node IDs.

This document includes the implementation of an algorithm to:

1. Detect conflicts dynamically in robot paths.

2. Resolve conflicts by stopping one of the robots based on priorities.

Backend Logic

```
import json


# Load robot paths
with open("robot_paths.json") as f:
    robot_paths = json.load(f)


# Load robot properties
with open("robot_properties.json") as f:
    robot_properties = json.load(f)


# Function to detect conflicts in robot paths
def detect_conflicts(paths):
    node_occupancy = {}
```

Fleet Manager Conflict Resolution Task

```
    conflicts = []

    for robot, path in paths.items():
        for node in path:
            if node not in node_occupancy:
                node_occupancy[node] = []
            node_occupancy[node].append(robot)

    for node, robots in node_occupancy.items():
        if len(robots) > 1:
            conflicts.append((node, robots))

    return conflicts

# Function to resolve conflicts dynamically
def resolve_conflicts(conflicts, properties):
    stop_instructions = {}

    for conflict in conflicts:
        node, robots = conflict
        sorted_robots = sorted(robots, key=lambda r: (
            properties[r]["priority"], -properties[r]["task_urgency"], -properties[r]["battery"]))
        for robot in sorted_robots[1:]:
            stop_instructions[robot] = f"Stop before reaching node {node}"
```

```
    return stop_instructions


# Main execution
conflicts = detect_conflicts(robot_paths)
stop_instructions = resolve_conflicts(conflicts, robot_properties)


# Save stop instructions to JSON file
with open("stop_instructions.json", "w") as f:
    json.dump(stop_instructions, f, indent=4)


print("Conflicts resolved. Stop instructions saved to 'stop_instructions.json'.")
```

Frontend Logic

```
const nodes = {
    1: { x: 50, y: 200 },
    2: { x: 150, y: 200 },
    3: { x: 250, y: 200 },
    4: { x: 350, y: 200 },
    5: { x: 450, y: 200 },
    6: { x: 50, y: 300 },
    7: { x: 150, y: 300 },
    8: { x: 350, y: 300 },
    9: { x: 450, y: 300 }
```

Fleet Manager Conflict Resolution Task

```
};


// Load robot paths and stop instructions
fetch("robots.json")
    .then((response) => response.json())
    .then((data) => startSimulation(data));


function startSimulation(data) {
    const { paths, stopInstructions } = data;


    // Create nodes
    const simulation = document.getElementById("simulation");
    for (let id in nodes) {
        const node = document.createElement("div");
        node.className = "node";
        node.style.left = `${nodes[id].x}px`;
        node.style.top = `${nodes[id].y}px`;
        simulation.appendChild(node);
    }


    // Create robots
    const robots = {};
    for (let robot in paths) {
        const robotDiv = document.createElement("div");
        robotDiv.className = `robot robot-${robot.toLowerCase().replace(" ", "-")}`;
```

Fleet Manager Conflict Resolution Task

```
    robots[robot] = { div: robotDiv, path: paths[robot], index: 0 };
    simulation.appendChild(robotDiv);
}

// Move robots
function moveRobots() {
    for (let robot in robots) {
        const { div, path, index } = robots[robot];
        if (index >= path.length) continue;

        const nextNode = path[index];
        if (stopInstructions[robot] && stopInstructions[robot].includes(nextNode)) {
            console.log(`${robot} is stopping at node ${nextNode}`);
            continue;
        }

        const { x, y } = nodes[nextNode];
        div.style.left = `${x - 10}px`;
        div.style.top = `${y - 10}px`;
        robots[robot].index++;
    }

    setTimeout(moveRobots, 1000);
}
```

```
    moveRobots();

}
```