VECROS ROBOTICS INTERN ASSIGNMENT

Name: Mohamed Samir EL-Sayed

Problem 1: Multi-Agent Pathfinding in 3D Grid

Problem Overview and Description

This problem is a common challenge in fields such as **robotics**, **logistics**, and **game development**. It involves finding the shortest paths for multiple agents (represented by the user-input sets of `{start, end}` points) in a 3D grid environment.

The key challenges are:

1. **Multiple Agents**: Since the problem involves two or more sets of `{start, end}` points, it is inherently a multi-agent pathfinding problem.
2. **Conflict Avoidance**: The paths of all agents must be conflict-free, meaning no two agents can occupy the same grid point at the same time.
3. **Obstacles**: The randomly assigned higher weights on certain grid points act as obstacles. The goal is to find the shortest path (i.e., the path with the smallest cumulative weight) while avoiding these obstacles.
4. **Time and Velocity**: Although the problem mentions velocity (`v m/s`), all agents share the same velocity in this scenario. Therefore, velocity is not a differentiating factor and can be ignored for simplicity. The focus is on ensuring that no conflicts occur at any time step.

Key Considerations

- **Conflict-Free Paths**: The primary challenge is to ensure that the computed paths for all agents do not intersect at any point in time. This means that at no time step should two agents occupy the same grid point.
- **Shortest Path**: The paths must not only be conflict-free but also optimal in terms of minimizing the cumulative weight (cost) of the path.
- **Visualization**: The final output includes a 3D plot of the computed paths, providing a clear and intuitive representation of the solution.

Simplified Problem Scope

- **Velocity**: Since all agents move at the same velocity, it does not affect the pathfinding logic and can be disregarded.
- **Time Steps**: The problem emphasizes avoiding conflicts at any time. However, in the implementation, we focus on ensuring that the final paths are conflict-free without explicitly modeling continuous time. Instead, we use discrete time steps corresponding to the movement of agents from one grid point to another.

Final Output

The solution involves:

1. Computing the shortest, conflict-free paths for all agents.
2. Visualizing the paths in a 3D plot, where each agent’s path is represented by a unique color.

Algorithm Overview:

The solution uses the Enhanced Conflict-Based Search (ECBS) algorithm, which extends the A* algorithm for multi-agent pathfinding. The grid is initialized with random weights assigned to some points, acting as obstacles. Each agent has a start and end position, and the goal is to compute the shortest path for each agent while ensuring no two agents occupy the same position at the same time. The A* algorithm is used to find the shortest path for individual agents, and ECBS resolves conflicts by adding constraints and re-planning paths until no conflicts remain. Finally, the paths are visualized in a 3D plot using Plotly, providing an intuitive representation of the solution.

Agent.py

Purpose: Represents an agent with a start and end position.

Attributes:

- start: The agent's starting position in the 3D grid.
- end: The agent's goal position in the 3D grid.
- path: Stores the computed path from start to end.
- constraints: Prevents the agent from occupying specific positions at specific times to avoid conflicts.

Grid.py

Purpose: Represents a 3D grid environment with randomly assigned weights.

Attributes:

- size: Size of the grid (e.g., 100 for a 100x100x100 grid).
- grid: A 3D numpy array storing weights for each grid point.

- **Methods**:
  - __init__: Initializes the grid and randomly assigns higher weights to some points.
    - weight_prob: Probability of a point having a high weight (default: 10%).
    - min_high_weight, max_high_weight: Range of high weights (default: 1 to 10).
  - get_weight: Returns the weight of a specific grid point.

Key Points:

1. **Grid Initialization**:
   - Creates a 3D grid of size (size + 1) x (size + 1) x (size + 1).
   - Randomly assigns higher weights to weight_prob * total_points grid points.
2. **Weights**:
   - Higher weights act as obstacles, making certain paths more costly.
3. **Utility**:
   - The get_weight method allows querying the weight of any grid point.

This class is essential for defining the environment in which agents navigate, providing the foundation for our pathfinding algorithms.

Utils.py

Purpose: Visualizes the paths of the multiple agents in a 3D plot using Plotly.

- **Key Features**:
    1. **Color-Coded Paths**:
        - Each agent's path is assigned a unique color from the `colors` list.
    2. **3D Visualization**:
        - The paths are plotted in a 3D space with `x`, `y`, and `z` axes.
    3. **Interactive Plot**:
        - The plot is interactive, allowing users to zoom, rotate, and pan.
    4. **Custom Layout**:
        - The layout includes axis labels, a title, and a legend for clarity.
        - The camera view is adjusted for better visualization.

How It Works:

1. **Input**:
    - A list of `Agent` objects, each containing a computed path (`agent.path`).
2. **Processing**:
    - Extracts the `x`, `y`, and `z` coordinates from each agent's path.
    - Assigns a unique color to each agent's path.
    - Adds each path as a trace to the 3D plot.
3. **Output**:
    - Displays an interactive 3D plot showing all agents' paths.

PathFinder.py

This file implements the core pathfinding algorithms: A* for single-agent pathfinding and Enhanced Conflict-Based Search (ECBS) for multi-agent pathfinding.

1. A* Algorithm (`a_star` method)

Overview

The **A* algorithm** is a heuristic search algorithm used to find the shortest path from a start node to a goal node in a graph or grid. It combines the cost to reach a node (`g_score`) with a heuristic estimate of the cost to the goal (`h_score`) to prioritize nodes in the search. The algorithm maintains an open set of nodes to explore, ordered by their total estimated cost (`f_score = g_score + h_score`). At each step, it expands the node with the lowest `f_score`, updates the costs of its neighbors, and continues until the goal is reached. A* is guaranteed to find the shortest path if the heuristic is admissible (never overestimates the cost). In my implementation, it is used to compute paths for individual agents while avoiding constraints.

Purpose:

Finds the shortest path for a single agent from `start` to `end` while avoiding constraints.

Steps:

1. **Initialization**:
    - `open_set`: A priority queue storing nodes to explore, prioritized by `f_score`.
    - `came_from`: Tracks the path by storing the parent of each node.
    - `g_score`: Stores the cost to reach each node from the start.
    - `constraints_set`: A set of constraints (position, time_step) to avoid.
2. **Exploration**:
    - Pop the node with the lowest `f_score` from `open_set`.
    - If the node is the goal, return the path.
    - Expand the node by generating its neighbors (6 directions in 3D).
    - For each neighbor:
        - Skip if it violates constraints or is outside the grid.
        - Calculate the tentative cost (`g_score`).

        - If the neighbor is not in `g_score` or the new cost is lower, update `g_score` and add it to `open_set`.

3. **Heuristic**:
    - Uses the **Manhattan distance** (`|x1 - x2| + |y1 - y2| + |z1 - z2|`) as the heuristic (`h`).
4. **Return**:
    - Returns the shortest path as a list of positions if a path is found.
    - Returns `None` if no path exists.

2. Enhanced Conflict-Based Search (ECBS) (`ecbs` method)

Overview

The Enhanced Conflict-Based Search (ECBS) algorithm extends A* to handle multi-agent pathfinding by resolving conflicts between agents' paths. It first computes initial paths for all agents using A*. Then, it detects conflicts where two agents occupy the same position at the same time step. For each conflict, it adds constraints to one of the agents to avoid the conflicting position and re-plans its path using A*. This process repeats until no conflicts remain. ECBS ensures that all agents reach their goals without collisions, making it suitable for multi-agent systems in environments like robotics, logistics, and game development.

Purpose:

Finds conflict-free paths for multiple agents using the A* algorithm and resolves conflicts iteratively.

Steps:

1. **Initial Path Planning**:
    - Compute initial paths for all agents using A* without constraints.
2. **Conflict Detection**:
    - Detect conflicts where two agents occupy the same position at the same time step.
3. **Conflict Resolution**:
    - For each conflict:
        - Add a constraint to one of the agents to avoid the conflicting position at the specific time step.
        - Re-plan the path for the constrained agent using A*.
    - Repeat until no conflicts remain.
4. **Return**:
    - Returns the list of agents with conflict-free paths.
    - Returns `None` if a conflict cannot be resolved.

---

3. Conflict Detection (`detect_conflicts` method)

Purpose:

Detects conflicts between agents' paths.

Steps:

1. **Compare Paths**:
    - For each pair of agents, compare their paths step by step.
    - If two agents occupy the same position at the same time step, mark it as a conflict.
2. **Return**:
    - Returns a list of conflicts, where each conflict is a tuple `(agent_idx, other_agent_idx, conflict_pos, conflict_step)`.

Multiple-Agent-path.py

This is the entry point of the program. It initializes the grid, agents, and pathfinder, computes conflict-free paths for all agents using the ECBS algorithm, and visualizes the results.

This file ties everything together, making it easy to run the multi-agent pathfinding system and visualize the results.

Output:

Running the following command:

python3 Question-1/Multiple-Agent-path.py

```
Grid size: (101, 101, 101)
Total grid points: 1030301
Number of high-weight points: 103030
Agent 1 path: [(0, 0, 0), (0, 0, 1), (1, 0, 1), (1, 0, 2), (1, 0, 3), (1, 0, 4), (1, 1, 4), (1, 2, 4), (1, 2, 5), (1, 2, 6), (1, 2, 7), (1, 3, 7)
, (1, 3, 8), (1, 3, 9), (1, 4, 9), (1, 4, 10), (1, 4, 11), (1, 5, 11), (1, 5, 12), (1, 5, 13), (1, 5, 14), (1, 5, 15), (1, 6, 15), (1, 6, 16), (1
, 6, 17), (1, 6, 18), (1, 6, 19), (1, 6, 20), (1, 6, 21), (1, 6, 22), (1, 6, 23), (1, 6, 24), (1, 6, 25), (1, 6, 26), (1, 6, 27), (1, 6, 28), (1,
 6, 29), (1, 7, 29), (1, 8, 29), (1, 8, 30), (1, 8, 31), (1, 8, 32), (1, 8, 33), (1, 8, 34), (1, 8, 35), (1, 8, 36), (1, 8, 37), (1, 9, 37), (1,
9, 38), (1, 9, 39), (1, 9, 40), (1, 9, 41), (1, 10, 41), (1, 10, 42), (1, 10, 43), (1, 10, 44), (1, 10, 45), (1, 10, 46), (2, 10, 46), (2, 10, 47
), (2, 10, 48), (2, 10, 49), (2, 10, 50), (2, 10, 51), (2, 11, 51), (2, 11, 52), (3, 11, 52), (3, 12, 52), (3, 13, 52), (3, 14, 52), (3, 15, 52),
 (3, 16, 52), (3, 17, 52), (3, 18, 52), (3, 19, 52), (3, 20, 52), (3, 21, 52), (3, 22, 52), (4, 22, 52), (4, 23, 52), (5, 23, 52), (5, 24, 52), (
5, 25, 52), (5, 26, 52), (5, 27, 52), (6, 27, 52), (7, 27, 52), (8, 27, 52), (8, 28, 52), (8, 29, 52), (8, 30, 52), (8, 31, 52), (8, 32, 52), (8,
 33, 52), (8, 34, 52), (8, 35, 52), (8, 36, 52), (8, 37, 52), (8, 38, 52), (8, 39, 52), (8, 40, 52), (9, 40, 52), (9, 41, 52), (9, 42, 52), (9, 4
3, 52), (9, 44, 52), (9, 45, 52), (9, 46, 52), (10, 46, 52), (11, 46, 52), (11, 47, 52), (11, 48, 52), (11, 49, 52), (11, 50, 52), (11, 51, 52),
(12, 51, 52), (13, 51, 52), (14, 51, 52), (14, 50, 52), (15, 50, 52), (16, 50, 52), (16, 51, 52), (16, 51, 53), (17, 51, 53), (18, 51, 53), (18,
51, 52), (19, 51, 52), (20, 51, 52), (21, 51, 52), (22, 51, 52), (23, 51, 52), (23, 50, 52), (24, 50, 52), (25, 50, 52), (25, 49, 52), (26, 49, 5
2), (27, 49, 52), (27, 50, 52), (28, 50, 52), (29, 50, 52), (30, 50, 52), (30, 51, 52), (31, 51, 52), (32, 51, 52), (33, 51, 52), (34, 51, 52), (
35, 51, 52), (35, 50, 52), (36, 50, 52), (37, 50, 52), (38, 50, 52), (38, 51, 52), (39, 51, 52), (40, 51, 52), (41, 51, 52), (42, 51, 52), (43, 5
1, 52), (44, 51, 52), (45, 51, 52), (46, 51, 52), (47, 51, 52), (48, 51, 52), (49, 51, 52), (50, 51, 52)]
```

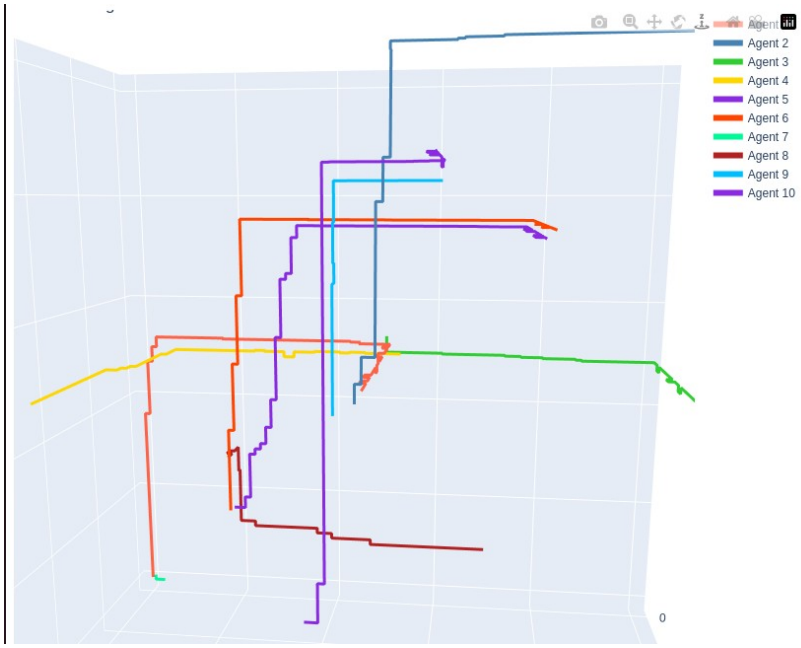
```
Agent 2 path: [(50, 50, 50), (50, 50, 51), (50, 50, 52), (50, 50, 53), (50, 50, 54), (50, 50, 55), (50, 50, 56), (50, 50, 57), (50, 50, 58), (50,
 50, 59), (50, 50, 60), (50, 50, 61), (50, 50, 62), (50, 51, 62), (50, 51, 63), (50, 51, 64), (50, 51, 65), (50, 51, 66), (50, 52, 66), (50, 52,
67), (50, 52, 68), (50, 52, 69), (50, 52, 70), (50, 52, 71), (50, 52, 72), (50, 52, 73), (50, 52, 74), (50, 52, 75), (50, 52, 76), (50, 52, 77),
(50, 52, 78), (50, 52, 79), (50, 52, 80), (50, 52, 81), (50, 52, 82), (50, 53, 82), (50, 53, 83), (50, 53, 84), (50, 53, 85), (50, 53, 86), (50,
53, 87), (50, 53, 88), (50, 53, 89), (50, 53, 90), (50, 53, 91), (50, 53, 92), (50, 53, 93), (50, 53, 94), (50, 53, 95), (50, 53, 96), (50, 53, 9
7), (50, 53, 98), (50, 54, 98), (50, 54, 99), (50, 54, 100), (50, 55, 100), (50, 56, 100), (50, 57, 100), (50, 58, 100), (50, 59, 100), (51, 59,
100), (51, 60, 100), (51, 61, 100), (51, 62, 100), (51, 63, 100), (51, 64, 100), (51, 65, 100), (51, 66, 100), (51, 67, 100), (52, 67, 100), (52,
 68, 100), (52, 69, 100), (52, 70, 100), (52, 71, 100), (53, 71, 100), (53, 72, 100), (53, 73, 100), (53, 74, 100), (53, 75, 100), (54, 75, 100),
 (54, 76, 100), (54, 77, 100), (54, 78, 100), (55, 78, 100), (55, 79, 100), (55, 80, 100), (55, 81, 100), (55, 82, 100), (55, 83, 100), (55, 84,
100), (55, 85, 100), (55, 86, 100), (55, 87, 100), (55, 88, 100), (55, 89, 100), (55, 90, 100), (56, 90, 100), (56, 91, 100), (57, 91, 100), (57,
 92, 100), (57, 93, 100), (57, 94, 100), (57, 95, 100), (57, 96, 100), (57, 97, 100), (57, 98, 100), (57, 99, 100), (57, 100, 100), (58, 100, 100
), (59, 100, 100), (60, 100, 100), (61, 100, 100), (62, 100, 100), (63, 100, 100), (64, 100, 100), (64, 99, 100), (65, 99, 100), (66, 99, 100), (
66, 100, 100), (67, 100, 100), (68, 100, 100), (69, 100, 100), (70, 100, 100), (71, 100, 100), (72, 100, 100), (73, 100, 100), (74, 100, 100), (7
4, 100, 99), (75, 100, 99), (76, 100, 99), (76, 100, 100), (77, 100, 100), (78, 100, 100), (79, 100, 100), (80, 100, 100), (81, 100, 100), (82, 1
00, 100), (83, 100, 100), (83, 99, 100), (84, 99, 100), (85, 99, 100), (85, 100, 100), (86, 100, 100), (87, 100, 100), (88, 100, 100), (89, 100,
100), (89, 99, 100), (90, 99, 100), (91, 99, 100), (91, 100, 100), (92, 100, 100), (93, 100, 100), (94, 100, 100), (95, 100, 100), (96, 100, 100)
, (97, 100, 100), (98, 100, 100), (99, 100, 100), (100, 100, 100)]
Agent 3 path: [(0, 50, 53), (0, 50, 52), (0, 50, 51), (0, 50, 50), (0, 51, 50), (0, 52, 50), (0, 53, 50), (0, 54, 50), (0, 55, 50), (1, 55, 50),
(1, 56, 50), (1, 57, 50), (1, 58, 50), (1, 59, 50), (1, 60, 50), (1, 61, 50), (1, 62, 50), (1, 63, 50), (1, 64, 50), (1, 65, 50), (1, 66, 50), (1
, 67, 50), (2, 67, 50), (2, 68, 50), (2, 69, 50), (2, 70, 50), (2, 71, 50), (2, 72, 50), (2, 73, 50), (2, 74, 50), (2, 75, 50), (2, 76, 50), (2,
77, 50), (2, 78, 50), (2, 79, 50), (2, 80, 50), (2, 81, 50), (2, 82, 50), (2, 83, 50), (2, 84, 50), (2, 85, 50), (2, 86, 50), (2, 87, 50), (2, 88
, 50), (2, 89, 50), (2, 90, 50), (2, 91, 50), (2, 92, 50), (2, 93, 50), (2, 94, 50), (2, 95, 50), (2, 96, 50), (2, 97, 50), (2, 98, 50), (2, 99,
50), (2, 100, 50), (3, 100, 50), (4, 100, 50), (5, 100, 50), (5, 99, 50), (6, 99, 50), (7, 99, 50), (7, 100, 50), (8, 100, 50), (9, 100, 50), (10
, 100, 50), (11, 100, 50), (12, 100, 50), (13, 100, 50), (14, 100, 50), (15, 100, 50), (16, 100, 50), (17, 100, 50), (18, 100, 50), (19, 100, 50)
, (20, 100, 50), (21, 100, 50), (22, 100, 50), (23, 100, 50), (24, 100, 50), (25, 100, 50), (26, 100, 50), (27, 100, 50), (28, 100, 50), (29, 100
, 50), (30, 100, 50), (31, 100, 50), (32, 100, 50), (33, 100, 50), (34, 100, 50), (35, 100, 50), (36, 100, 50), (37, 100, 50), (38, 100, 50), (39
, 100, 50), (40, 100, 50), (41, 100, 50), (42, 100, 50), (43, 100, 50), (44, 100, 50), (45, 100, 50), (46, 100, 50), (47, 100, 50), (48, 100, 50)
, (49, 100, 50), (49, 99, 50), (50, 99, 50), (51, 99, 50), (51, 100, 50), (52, 100, 50)]
```

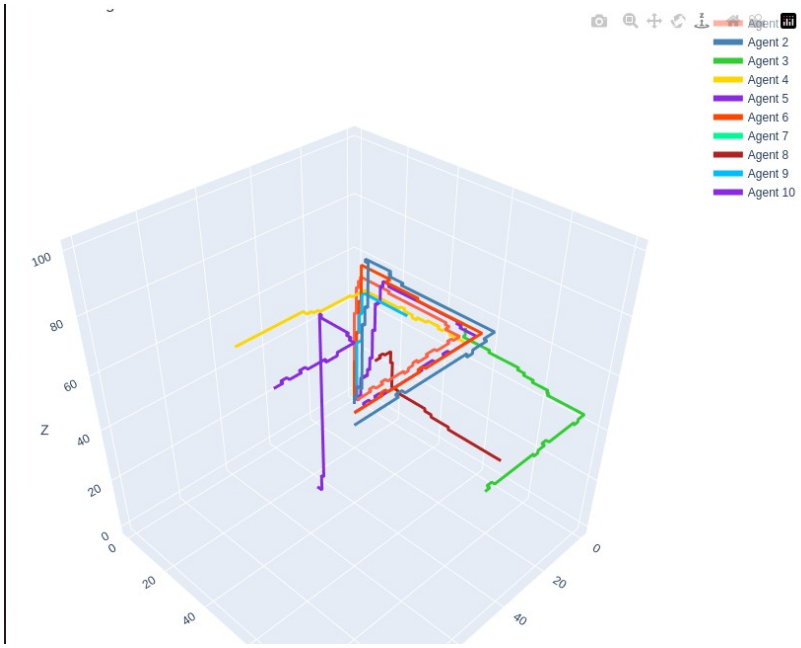
```
Agent 4 path: [(56, 3, 50), (55, 3, 50), (54, 3, 50), (53, 3, 50), (52, 3, 50), (51, 3, 50), (50, 3, 50), (49, 3, 50), (48, 3, 50), (47, 3, 50),
(47, 4, 50), (46, 4, 50), (45, 4, 50), (44, 4, 50), (43, 4, 50), (42, 4, 50), (41, 4, 50), (40, 4, 50), (39, 4, 50), (38, 4, 50), (38, 5, 50), (3
7, 5, 50), (37, 6, 50), (36, 6, 50), (35, 6, 50), (34, 6, 50), (33, 6, 50), (33, 7, 50), (32, 7, 50), (31, 7, 50), (30, 7, 50), (29, 7, 50), (28,
 7, 50), (27, 7, 50), (26, 7, 50), (26, 8, 50), (25, 8, 50), (24, 8, 50), (23, 8, 50), (22, 8, 50), (21, 8, 50), (20, 8, 50), (20, 9, 50), (19, 9
, 50), (18, 9, 50), (17, 9, 50), (16, 9, 50), (15, 9, 50), (14, 9, 50), (13, 9, 50), (12, 9, 50), (11, 9, 50), (10, 9, 50), (9, 9, 50), (9, 10, 5
0), (8, 10, 50), (7, 10, 50), (6, 10, 50), (5, 10, 50), (4, 10, 50), (3, 10, 50), (3, 11, 50), (3, 12, 50), (3, 13, 50), (3, 14, 50), (3, 15, 50)
, (3, 16, 50), (3, 17, 50), (3, 18, 50), (3, 19, 50), (3, 20, 50), (3, 21, 50), (3, 22, 50), (3, 23, 50), (3, 24, 50), (3, 25, 50), (3, 26, 50),
(3, 27, 50), (2, 27, 50), (2, 28, 50), (1, 28, 50), (1, 29, 50), (1, 30, 50), (1, 31, 50), (1, 32, 50), (1, 33, 50), (1, 34, 50), (1, 35, 50), (1
, 36, 50), (1, 37, 50), (1, 38, 50), (1, 39, 50), (1, 40, 50), (1, 41, 50), (1, 42, 50), (1, 43, 50), (1, 44, 50), (1, 45, 50), (1, 46, 50), (1,
47, 50), (1, 48, 50), (1, 49, 50), (1, 50, 50), (2, 50, 50), (2, 51, 50), (2, 52, 50), (2, 53, 50), (3, 53, 50)]
Agent 5 path: [(25, 25, 25), (25, 25, 26), (25, 26, 26), (25, 26, 27), (25, 26, 28), (25, 26, 29), (25, 27, 29), (25, 27, 30), (25, 27, 31), (25,
 27, 32), (25, 27, 33), (25, 27, 34), (25, 27, 35), (25, 27, 36), (25, 27, 37), (25, 27, 38), (25, 27, 39), (25, 27, 40), (25, 27, 41), (25, 27,
42), (25, 27, 43), (25, 27, 44), (25, 27, 45), (25, 27, 46), (25, 27, 47), (25, 28, 47), (25, 28, 48), (25, 28, 49), (25, 29, 49), (25, 30, 49),
(25, 30, 50), (25, 30, 51), (25, 30, 52), (25, 30, 53), (25, 30, 54), (25, 30, 55), (25, 30, 56), (25, 30, 57), (25, 30, 58), (25, 30, 59), (25,
30, 60), (25, 30, 61), (25, 31, 61), (25, 31, 62), (25, 31, 63), (25, 31, 64), (25, 31, 65), (25, 31, 66), (25, 31, 67), (25, 31, 68), (25, 31, 6
9), (25, 31, 70), (25, 31, 71), (25, 31, 72), (25, 31, 73), (25, 31, 74), (25, 31, 75), (25, 32, 75), (26, 32, 75), (26, 33, 75), (26, 34, 75), (
26, 35, 75), (26, 36, 75), (26, 37, 75), (26, 38, 75), (26, 39, 75), (26, 40, 75), (26, 41, 75), (26, 42, 75), (26, 43, 75), (26, 44, 75), (26, 4
5, 75), (26, 46, 75), (27, 46, 75), (27, 47, 75), (27, 48, 75), (27, 49, 75), (27, 50, 75), (28, 50, 75), (28, 51, 75), (28, 52, 75), (28, 53, 75
), (28, 54, 75), (29, 54, 75), (29, 55, 75), (30, 55, 75), (31, 55, 75), (32, 55, 75), (33, 55, 75), (34, 55, 75), (34, 56, 75), (34, 57, 75), (3
4, 58, 75), (35, 58, 75), (35, 59, 75), (35, 60, 75), (35, 61, 75), (35, 62, 75), (35, 63, 75), (35, 64, 75), (35, 65, 75), (35, 66, 75), (35, 67
, 75), (35, 68, 75), (35, 69, 75), (35, 70, 75), (35, 71, 75), (35, 72, 75), (35, 73, 75), (35, 74, 75), (35, 75, 75), (36, 75, 75), (37, 75, 75)
, (38, 75, 75), (39, 75, 75), (40, 75, 75), (41, 75, 75), (42, 75, 75), (43, 75, 75), (44, 75, 75), (45, 75, 75), (46, 75, 75), (47, 75, 75), (48
, 75, 75), (49, 75, 75), (50, 75, 75), (51, 75, 75), (52, 75, 75), (53, 75, 75), (54, 75, 75), (55, 75, 75), (56, 75, 75), (57, 75, 75), (58, 75,
 75), (59, 75, 75), (60, 75, 75), (61, 75, 75), (62, 75, 75), (63, 75, 75), (64, 75, 75), (65, 75, 75), (66, 75, 75), (67, 75, 75), (68, 75, 75),
 (69, 75, 75), (69, 75, 74), (70, 75, 74), (71, 75, 74), (71, 75, 75), (72, 75, 75), (73, 75, 75), (74, 75, 75), (75, 75, 75)]
Agent 6 path: [(24, 24, 24), (24, 24, 25), (24, 24, 26), (24, 25, 26), (25, 25, 26), (25, 26, 26), (25, 26, 27), (25, 26, 28), (25, 26, 29), (25,
 27, 29), (25, 27, 30), (25, 27, 31), (25, 27, 32), (25, 27, 33), (25, 27, 34), (25, 27, 35), (25, 27, 36), (25, 27, 37), (25, 27, 38), (25, 27,
39), (25, 27, 40), (25, 27, 41), (25, 27, 42), (25, 27, 43), (25, 27, 44), (25, 27, 45), (25, 27, 46), (25, 27, 47), (25, 28, 47), (25, 28, 48),
(25, 28, 49), (25, 29, 49), (25, 30, 49), (25, 30, 50), (25, 30, 51), (25, 30, 52), (25, 30, 53), (25, 30, 54), (25, 30, 55), (25, 30, 56), (25,
30, 57), (25, 30, 58), (25, 30, 59), (25, 30, 60), (25, 30, 61), (25, 31, 61), (25, 31, 62), (25, 31, 63), (25, 31, 64), (25, 31, 65), (25, 31, 6
6), (25, 31, 67), (25, 31, 68), (25, 31, 69), (25, 31, 70), (25, 31, 71), (25, 31, 72), (25, 31, 73), (25, 31, 74), (25, 31, 75), (25, 31, 76), (
```

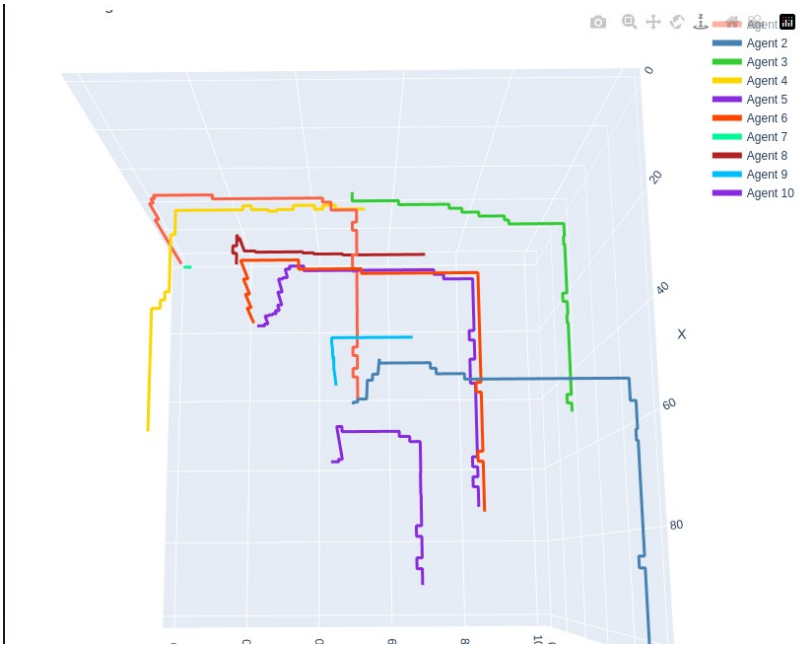
The program outputs the following:

1. **Grid Information**:
    - Size of the grid (`101x101x101`).
    - Total number of grid points (`1,030,301`).
    - Number of high-weight points (`103,030`), which act as obstacles.
2. **Agent Paths**:
    - Each agent's path is printed as a sequence of grid points from its start to its end position.
    - For example, Agent 1 moves from `(0, 0, 0)` to `(50, 51, 52)`.
3. **Visualization**:
    - The text output alone is not sufficient to understand the paths in a 3D space.
    - The **3D visualization** provides an interactive and intuitive representation of the paths, making it easier to analyze and verify the results.

Visualization
It may not be well shown here as a 3d plot is hard to be shown in a 2d but I recommend run the file and check the 3D plot so you can see how there is zero conflict in it






Agent 1
Agent 2
Agent 3
Agent 4
Agent 5
Agent 6
Agent 7
Agent 8
Agent 9
Agent 10
0
20
40
X
60
80

Problem 2: Drone Simulation and Control

Problem Overview and Description

This problem involves planning a mission for a quadcopter using a predefined set of 15 waypoints, each containing latitude (`lat`), longitude (`lon`), and altitude (`alt`) values. The mission is executed in auto mode using Dronekit or pymavlink, with the drone landing at the last waypoint. After reaching the 10th waypoint, a new waypoint is added to the mission, positioned 100 meters perpendicular to the current direction of travel, and the drone continues with the updated mission. Throughout the mission, the estimated time and distance to completion are printed at every instance. Finally, the path of travel is plotted in 2D to visualize the mission.

Algorithm Overview

1. Setup and Mock Vehicle Connection

- **Define `connect_mock()` Function:**
    - Creates a `MockVehicle` class to simulate a drone vehicle.
    - The mock vehicle has methods to mimic vehicle behavior such as arming, taking off, moving to locations, and managing missions.

2. Waypoint Generation and Perpendicular Point Calculation

- **Define `generate_waypoints()` Function:**
    - Generates a list of waypoints starting from a home location.
    - Introduces random perturbations in latitude and longitude.
    - Sets an altitude of 30 meters for all waypoints except the last one, which is set to 0 meters.
- **Define `perpendicular_point()` Function:**
    - Calculates a point perpendicular to the line between two given points.
    - Uses bearing calculations to determine the new point at a specified distance.
- **Define `haversine()` Function:**
    - Computes the distance between two points on the Earth's surface using the Haversine formula.

3. Mission Creation and Management

- **Define `Mission` Class:**
    - Initializes with a home location and a number of waypoints.
    - Uses `generate_waypoints()` to create a list of waypoints.
    - Creates mission commands for each waypoint using `dronekit.Command`.
- **Add Waypoint Method:**
    - Inserts a new waypoint into the mission at a specified index.
    - Updates the mission commands accordingly.

4. Main Execution Flow

- **Connect to Mock Vehicle:**
    - Simulates connecting to a drone vehicle.
    - Sets the home location.
- **Create and Upload Mission:**
    - Initializes a `Mission` object with the home location.
    - Uploads the mission to the mock vehicle.
- **Arm and Takeoff:**
    - Arms the vehicle and takes off to the altitude of the first waypoint.
- **Simulate Mission Execution:**
    - Switches the vehicle to AUTO mode to start the mission.
    - Iterates through the waypoints, simulating reaching each one.
    - Updates the vehicle's location and next waypoint index.
    - Calculates and prints the estimated time and distance remaining to complete the mission.
- **Dynamic Waypoint Addition:**
    - After reaching the 10th waypoint, adds a new waypoint perpendicular to the line between the 8th and 9th waypoints.
    - Updates the mission with the new waypoint.
- **Simulate Time Delay:**
    - Uses `time.sleep(1)` to simulate the passage of time between waypoints.
- **Mission Completion:**
    - Prints a completion message when all waypoints are reached.
- **Plot Flight Path:**
    - Uses `matplotlib.pyplot` to plot the latitude and longitude of the waypoints.
    - Displays the flight path as a route on a 2D plot.
- **Close Vehicle Connection:**
    - Simulates closing the connection to the vehicle.

Code Exploration

connect_mock() Function

The `connect_mock()` function creates a simulated drone, or "mock vehicle," which allows for the testing of mission planning and execution logic without the need for a physical drone. This mock vehicle, defined within the function, has attributes and methods that mimic the behavior of a real drone, such as arming, taking off, navigating to waypoints, and managing mission commands. It initializes with a default location and mode, and provides functionalities to update mission plans, track mission progress, and simulate communication with the drone

generate_waypoints()

This function is used to create a realistic flight path for the drone mission. The perturbations in latitude and longitude simulate real-world GPS inaccuracies or environmental factors, while the linear progression ensures the drone moves in a general direction. The fixed altitude (except for the last waypoint) ensures the drone maintains a consistent height during the mission, with a safe landing at the end.

perpendicular_point()

The `perpendicular_point(p1, p2, distance=100)` function calculates a new geographic point that is perpendicular to the line connecting two given points (`p1` and `p2`) at a specified distance. It uses the latitude and longitude of the input points to compute the direction (bearing) between them, then calculates a new bearing that is 90 degrees offset from the original. Using the Earth's radius and trigonometric formulas, it determines the coordinates of the new point at the specified perpendicular distance, ensuring the calculation accounts for the Earth's curvature. The function returns the new point with the same altitude as `p2`, allowing for dynamic adjustments to a drone's flight path, such as creating detours or avoiding obstacles during a mission.

`haversine()`

The `haversine()` function calculates the great-circle distance between two points on the Earth's surface, given their latitude and longitude in degrees. This is based on the **Haversine formula**, which accounts for the Earth's curvature to provide an accurate distance measurement

The function is used to calculate the distance between two geographic coordinates, which is essential for tracking the drone's progress during a mission. It helps estimate the remaining distance to the next waypoint, total mission distance, and mission time based on the drone's speed. By using the Haversine formula, the function ensures accurate distance calculations, accounting for the Earth's spherical shape.

Mission class

The `Mission` class manages the planning and execution of our drone mission by generating a sequence of waypoints starting from a specified home location and converting them into MAVLink commands that the drone can execute. During initialization, it uses the `generate_waypoints()` function to create a list of waypoints with slight perturbations for realism and the `create_commands()` method to convert these waypoints into MAVLink-compatible commands formatted as `MAV_CMD_NAV_WAYPOINT`. The class also provides the `add_waypoint()` method to dynamically insert new waypoints into the mission at a specified index, allowing for real-time adjustments such as detours or obstacle avoidance. This class serves as the core component for the mission planning, enabling our drone to follow a predefined path while supporting flexibility for in-flight modifications.

Main()

The `main()` function simulates the complete drone mission from start to finish, beginning with connecting to the mock vehicle and defining a home location.

It creates the mission with a series of waypoints, uploads the mission to the vehicle, and simulates arming, taking off, and switching to AUTO mode to execute the mission.

As the drone progresses through each waypoint, the function calculates the mission's elapsed time and remaining distance using the Haversine formula, assuming a constant speed of 5 m/s.

After reaching the 10th waypoint, the new perpendicular waypoint is dynamically added to the mission, and the updated commands are uploaded to the vehicle. The mission concludes by plotting the flight path using `matplotlib` and closing the vehicle connection. This function encapsulates the entire mission lifecycle, including planning, execution, real-time adjustments, and visualization, providing a comprehensive simulation of the drone operations.

Output Sample

```
mohamed:~/Vecros$ python3 Question-2/Drone.py
Connecting to vehicle...
Connected to vehicle!
Mission uploaded.
Motors armed.
Taking off to 30 meters.
Switched to AUTO mode.
Reached waypoint 1: Lat=37.77491718866227, Lon=-122.41936547536858, Alt=30
Going to: Lat=37.77491718866227, Lon=-122.41936547536858, Alt=30
Estimated time to complete mission: 40.95961141808213 seconds
Estimated distance to complete mission: 204.79805709041068 meters


Reached waypoint 2: Lat=37.77504422303845, Lon=-122.41932324782019, Alt=30
Going to: Lat=37.77504422303845, Lon=-122.41932324782019, Alt=30
Estimated time to complete mission: 40.95961141808214 seconds
Estimated distance to complete mission: 204.7980570904107 meters


Reached waypoint 3: Lat=37.775109504582204, Lon=-122.41923509854337, Alt=30
Going to: Lat=37.775109504582204, Lon=-122.41923509854337, Alt=30
Estimated time to complete mission: 40.95961141808213 seconds
Estimated distance to complete mission: 204.79805709041068 meters
```

```


Reached waypoint 5: Lat=37.775347517256044, Lon=-122.41901083543503, Alt=30
Going to: Lat=37.775347517256044, Lon=-122.41901083543503, Alt=30
Estimated time to complete mission: 40.95961141808213 seconds
Estimated distance to complete mission: 204.79805709041068 meters


Reached waypoint 6: Lat=37.775447568246136, Lon=-122.41893737776022, Alt=30
Going to: Lat=37.775447568246136, Lon=-122.41893737776022, Alt=30
Estimated time to complete mission: 40.95961141808213 seconds
Estimated distance to complete mission: 204.79805709041068 meters


Reached waypoint 7: Lat=37.775487725695726, Lon=-122.41883219143985, Alt=30
Going to: Lat=37.775487725695726, Lon=-122.41883219143985, Alt=30
Estimated time to complete mission: 40.95961141808214 seconds
Estimated distance to complete mission: 204.79805709041068 meters


Reached waypoint 8: Lat=37.77555838819003, Lon=-122.4187479858, Alt=30
Going to: Lat=37.77555838819003, Lon=-122.4187479858, Alt=30
Estimated time to complete mission: 40.95961141808213 seconds
Estimated distance to complete mission: 204.79805709041068 meters
```

```
Reached waypoint 9: Lat=37.775653690642116, Lon=-122.41858235813174, Alt=30
Going to: Lat=37.775653690642116, Lon=-122.41858235813174, Alt=30
Estimated time to complete mission: 40.95961141808213 seconds
Estimated distance to complete mission: 204.79805709041068 meters


Reached waypoint 10: Lat=37.77579010305008, Lon=-122.41850362516803, Alt=30
Going to: Lat=37.77579010305008, Lon=-122.41850362516803, Alt=30
Estimated time to complete mission: 40.95961141808213 seconds
Estimated distance to complete mission: 204.79805709041068 meters


New waypoint added at position 10: Lat=37.77541683361801, Lon=-122.41746847546212, Alt=30


Reached waypoint 11: Lat=37.77541683361801, Lon=-122.41746847546212, Alt=30
Going to: Lat=37.77541683361801, Lon=-122.41746847546212, Alt=30
Estimated time to complete mission: 77.44493653869607 seconds
Estimated distance to complete mission: 387.2246826934803 meters


Reached waypoint 12: Lat=37.775941158800684, Lon=-122.41840913157922, Alt=30
Going to: Lat=37.775941158800684, Lon=-122.41840913157922, Alt=30
Estimated time to complete mission: 77.44493653869607 seconds
```