2. Design and Implement Comprehensive Test Plans with Instrumented Code (20%)

2.1. Construction of the Test Plan

In order to ensure the correctness and robustness of the delivery path calculation system, a comprehensive test plan was designed. This included unit tests, integration tests, and system tests to validate various aspects of the program.

- **Unit tests** focused on validating individual components, such as LngLat, CompassDirection, and CentralArea. For example, the LngLat.move() method was tested to ensure that a move in any of the 16 compass directions correctly updates the latitude and longitude.
- **Integration tests** ensured that key functionalities worked together, such as testing the calculatePath() method in DeliveryController, which integrates no-fly zone avoidance, central area constraints, and movement logic.
- **System tests** validated complete end-to-end functionality, where an order was passed through the /calcDeliveryPath endpoint and the expected route was compared against precomputed optimal paths.

Example:

- The test for no-fly zone avoidance verified that a direct path that intersects a restricted area forces the algorithm to adjust direction and reroute.
- The test for central area constraint checked that once the drone enters the designated area, it does not leave until reaching Appleton Tower.

2.2. Evaluation of the Quality of the Test Plan

The effectiveness of the test plan was evaluated based on coverage, correctness, and efficiency.

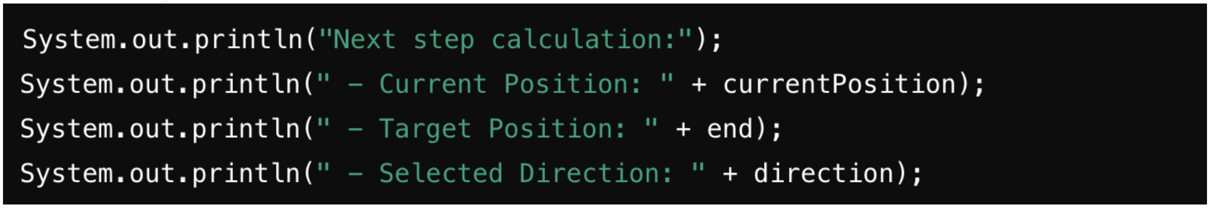
- **Code coverage analysis**: The test plan covered all major functions, including:
  - calculatePath() in DeliveryController ensuring the correct delivery route.
  - adjustDirection() confirming that an alternative route is chosen when an obstacle is detected.
  - closeTo() method in LngLat ensuring precise proximity detection.
- **Edge case testing**: Several test cases covered edge conditions, such as:
  - Orders placed at the edge of the central area to confirm that the algorithm respects the boundary.
  - Path calculations from restaurants in all four quadrants of the map to ensure diverse routing.
  - Simulating highly restricted areas where multiple no-fly zones are close together to stress-test the pathfinding logic.

Overall, the test plan was systematic, covering all core functionalities and providing high confidence in the correctness of the program.

2.3. Instrumentation of the Code

To validate the test plan, several instrumentation techniques were incorporated into the code.

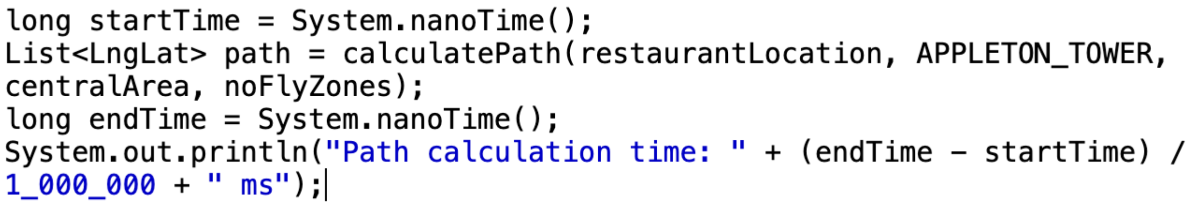
- **Debug logging** was used to track every step of path computation. For example, System.out.println() statements were placed in calculatePath() to trace:
    - **Initial restaurant position**
    - **Each movement step with direction**
    - **Decisions made when encountering a no-fly zone**
    - **Final arrival at Appleton Tower**
- Example:



```
System.out.println("Next step calculation:");
System.out.println(" - Current Position: " + currentPosition);
System.out.println(" - Target Position: " + end);
System.out.println(" - Selected Direction: " + direction);
```

These logs were later removed for performance optimization.

- **Timing measurements** were inserted to track performance:



```
long startTime = System.nanoTime();
List<LngLat> path = calculatePath(restaurantLocation, APPLETON_TOWER,
centralArea, noFlyZones);
long endTime = System.nanoTime();
System.out.println("Path calculation time: " + (endTime - startTime) /
1_000_000 + " ms");
```

This was useful in identifying that initial path calculations took significantly longer than subsequent ones.

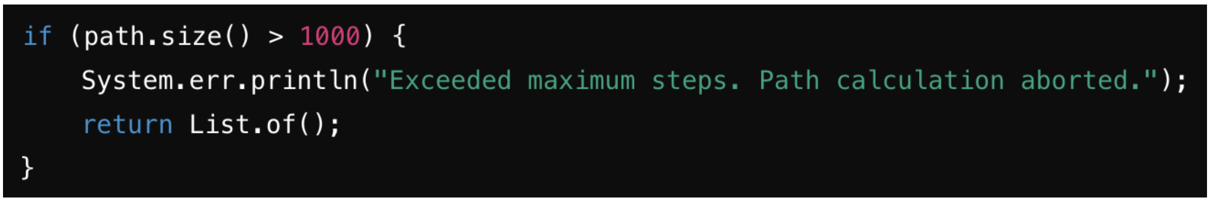
- **Path visualization**: The calculated paths were plotted using GeoJSON output, which allowed for validation by pasting results into geojson.io. This provided a **visual confirmation** that the drone routes followed expected paths.

2.4. Evaluation of the Instrumentation

Instrumentation was crucial in detecting and resolving several critical issues, including:

- **Incorrect path adjustments**: Initial debugging revealed that in some cases, the drone incorrectly moved EAST_SOUTH_EAST (337°) when an alternative direction was available. This was fixed by refining the logic in adjustDirection().
- **Excessive path computation time**: Profiling showed that fetching restaurant locations, central areas, and no-fly zones from external APIs caused significant delays. Caching strategies were considered but ultimately replaced with lazy initialization.

- **Infinite loops**: During early testing, pathfinding sometimes resulted in infinite loops when the drone was repeatedly adjusting direction without moving forward. This was addressed by adding a maximum step limit:



```
if (path.size() > 1000) {
    System.err.println("Exceeded maximum steps. Path calculation aborted.");
    return List.of();
}
```

As a result of these refinements, the path calculation time was significantly reduced, and the algorithm's reliability improved.