Testing Document

This document provides evidence of applied testing techniques, coverage, and results. I will cover each of the tests needed to satisfy the three main requirements I outlined in the **Test Planning Document**. Throughout the document I will reference the **Test Cases** IDs so that each test can be linked to their requirement and evidence in other documents.

R1: Order Validation

The system must be able to correctly identify invalid orders and return the correct validation error code and status code 400. The system must be able to identify valid orders and return status code 200.

Requirements for Credit Card Information:

- FR-OV-01: Expiry date must be in the future
- FR-OV-02: Card number must be numeric and be 16 digits
- FR-OV-03: CVV number must be numeric and 3 digits

Requirements for Order:

- FR-OV-04: Reject orders containing zero pizzas
- FR-OV-05: Reject orders exceeding the maximum pizza count of 4
- FR-OV-06: Reject orders containing pizzas that do not exist in any restaurant
- FR-OV-07: Reject orders containing pizzas from multiple restaurants
- FR-OV-08: Reject orders with incorrect total price (price of pizzas plus delivery fee)
- FR-OV-09: Reject orders placed when the restaurant is closed

Testing Techniques

For this requirement, I used black-box functional specification-based testing as the ILP coursework has very clear and specific guidelines that order validation must follow. As the ILP auto-marker is structured to test each requirement such that only a single requirement fails and causes an order to be invalid, I supported the specification-based testing with negative testing.

Equivalence partitioning was used to test valid and invalid cases for each requirement, further tested by using boundary values analysis. Evidence and description of each test is covered in more depth in the **Test Cases** document.

Adequacy

These techniques combine to create a full coverage of the errors that can arise in order validation because faults are highly concentrated at boundaries and invalid partitions. By isolating and rigorously testing each possible failure condition, test suite coverage can be extensive. All validation codes are tested at least once and both valid and invalid cases are tested therefore I can conclude that the Order Validation requirement has been extensively tested.

R2: Flight Path Generation

The system must generate a valid delivery path between the restaurant and Appleton Tower while respecting spatial constraints (e.g. no-fly zones) defined in the ILP specification. For valid orders, a sequence of positions representing the delivery path must be returned. If no valid path exists, the system must return an HTTP 400 response.

Requirements:

FR-N-04: The system will avoid entering no-fly zones

FR-N-05: Once the drone enters the Central Area, it will not leave it again

FR-N-06: Return HTTP 400 if no valid delivery path can be found [not applicable to the ILP specification]

FR-N-07: For valid orders, the system will return a sequence of positions representing the delivery path

FR-N-08: The GeoJSON endpoint shall return the same path as the delivery path endpoint, excluding hover steps [not explicitly tested due to resource constraints]

Flight Path Generation is the most complicated of the requirements to tests since there are infinitely many coordinates for the algorithm to choose from to construct the shortest path. Hence, keeping in mind the limited resources and need for prioritisation of requirements discussed in the **Test Planning Document**, I will focus on testing the validity and existence of the flight path rather than exhaustive optimality proofs of shortest-path behaviour. This would likely make a difference of mere seconds in the simulation and therefore does not justify the effort within the constraints of the ILP coursework.

Therefore, we can split R2 into 2 main requirements:

R2.1 – A valid flight path is generated when the conditions are feasible (which is always assumed to be under the ILP specification) [FR-N-06, FR-N-07]

R2.2 – The generated flight path obeys no-fly zones and Central Area restrictions [FR-N-04, FR-N-05]

Testing Techniques

These requirements are inherently integration heavy as they depend on coordinated behaviour between routing logic, geometric calculations, region definitions, and REST endpoint handling.

Testing for flight path generation combines black-box functional specification-based testing for externally observable behaviour, property-based reasoning for path properties, and integration testing. Integration testing is used as routing behaviour relies on the interaction of multiple components and classes such as restaurant data, central area and no-fly zone constraints and further geometric drone movement specification rules.

FR-N-07 is the only requirement that is suitable for unit testing. I created a unit test to generate the path for each restaurant given a valid order in addition to a few with invalid orders for negative testing.

Adequacy

Adequacy was assessed by ensuring that all routing constraints and both valid and invalid path cases are tested. Path legality is verified through a combination of automated assertions and manual inspection.

This mixed approach is appropriate given the complexity of the requirement and the limited feasibility of fully automated validation. However, in the hypothetical deployment of the drone service, this requirement would be given the most resources and test coverage would be in more depth; testing techniques like catalogue-based and category-partition testing would be appropriate.

R3: Drone Movement

The drone must move only in the 16 compass directions and with a set step size and proximity threshold. This requirement focuses on correctness of individual movement steps and coordinate handling.

Requirements for movement:

FR-N-01: Move the drone in the 16 discrete compass directions
FR-N-02: Move the drone by a fixed distance per step
FR-N-03: The drone will hover for one move above the restaurant or Appleton Tower when collecting or delivering an order

Requirements for coordinates:

UT-GEO-01: Reject coordinates outside valid latitude/longitude bounds
UT-GEO-02: Return zero distance for identical coordinates
Be symmetric with respect to argument order
UT-GEO-03: Return true when coordinate differences are less than 0.00015

Testing Techniques

These requirements are low-level, deterministic, and independent of REST or routing logic and therefore can be tested primarily with unit testing and structural white-box testing.

Unit tests were used to validate step size and direction for all 16 compass directions as well as handling LngLat coordinate boundaries and behaviour to ensure full coverage of the discrete movement logic. This prevents untested directional cases from causing subtle routing failures. Furthermore, unit tests were constructed to access the mathematical correctness of distance/proximity calculations.

This level of testing allows precise fault localisation and avoids masking numerical errors within higher-level routing behaviour.

Hover behaviour (FR-N-03) was validated indirectly through integration tests and informally through manual inspection of the given coordinate list of the flight path, as it only manifests meaningfully during order collection and delivery.

Adequacy

Adequacy was assessed by ensuring that all movement rules are tested deterministically, numerical thresholds, such as proximity thresholds, are tested explicitly, and movement logic is validated independently of the rest of the system. This provides strong confidence in the correctness of the foundational geometry used throughout the system.