Test Plan

1-Introduction

This test plan outlines the methods outlines how the PizzaDronz project will be tested to align with the requirements that have been listed in the LO1 Requirements Document. This project was completed in the 2023-2024 Semester 1 and so the tests are being created after the completion of the project and the takedown of the corresponding REST server and so all testing will be done in sequential order without gaps for refinement of the projects base code.

2-Priorities of requirements

Only the core requirements are listed here, there are many others in the LO1 document, but not all are of importance enough to require a detailed breakdown.

2.1 R1: The drone must never enter the no-fly zones

This is the highest priority requirement due to the safety and noise concerns being a serious concern. While other faults such as possible poor routing could also be a concern, safety and respecting the regulations on drone flight in these densely populated areas must take the focus in testing.

Decomposition of this requirement would give us R1.1: No "Move" can have a start-point or endpoint inside a no-fly zone, R1.2: No "Node" inside a no-fly zone can be valid, R1.3: The A Star algorithm must never produce a flight path with a LngLat position inside a no-fly zone.

Validation and verification

R1.1 – The "Move" class should be tested to ensure that it has inbuilt limitations on its constructor to not allow points with a start or endpoint inside a no-fly zone and this can be checked by attempts to create such Illegal move instances

R1.2 – The "Node" class should be tested similarly to the "Move" class

R1.3 – The A Star algorithm class "FlightAStar" should be tested to ensure that invalid Nodes are removed and cannot be used in the generated Path this can be checked by trying to generate paths that must circumvent No-fly zones and ensure that the path is valid.

2.2 R2: The orders for the day must be validated

The orders for the day must be validated in accordance with the specification e.g. valid credit card expiry etc. this is to ensure that only valid orders receive flightpaths, and the drone is not flying to a restaurant that had an invalid order and thus no pizza to deliver which would waste time.

Decomposition of this requirement would give us R2.1: All orders must be validated and assigned the corresponding correct OrderValidationCode, R2.2: Invalid orders must not receive flightpaths, R2.3: All orders for the day must be retrieved from the REST server.

Validation and verification

R2.1 – OrderValidator class should be tested with a range of valid and invalid Order objects to ensure that each is correctly assigned a OrderValidationCode by comparison.

R2.2- OrderValidator and the Main function should be tested in tandem with some lists of valid and invalid orders to ensure that every time the flightpath generated does not include any invalid orders.

R2.3 – Orders should be retrieved and then tested against the data in the rest server to ensure that all the data has correctly been received.

2.3 R3: The files generated should be in the correct format

The files generated should be in:

deliveries-YYYY-MM-DD.json: Records all orders with status and cost.

flightpath-YYYY-MM-DD.json: Logs every drone move with coordinates and timestamps.

drone-YYYY-MM-DD.geojson: Visualizes the flight path in GeoJSON format.

Decomposition of this requirement would give us R3.1: deliveries file must be in correct format, R3.2: flightpath file must be in correct format, R3.3: drone file must be in correct format.

Validation and verification

R3.1: The FileHandler class should be tested with the method recordDeliveries in particular with a data set of deliveries to ensure that the created files stick to the designated naming conventions and the data inside is in correct format.

R3.2: Similarly to 3.1 but looking at the recordMove method in FileHandler.

R3.3 Similarly to 3.2 but looking at the recordGeoJson method in FileHandler.

2.4 R4: The application should complete running in under 60 seconds.

The overall process of getting the data from the REST server, validating it, generating flightpaths and then files being generated needs to complete within this timeframe as the specifications state.

Decomposition of this process is not needed as it encapsulates the whole program, and it is a testable metric

Validation and verification

R4: The program should be tested on many different dates holding many different orders to ensure that on general it can run in this timeframe, obviously it cannot always run under 60 seconds as if there are an abnormal number of orders on a day it might take longer so a range of orders within a standard deviation of the mean should be tested.

3-Scaffolding

R1: generating test data for points inside the no flight zones will be required but this generated test data can be used across all the decomposed parts of R1, manual inspections of flightpaths created by the A Star algorithm will not require any scaffolding as this can be done in GeoJson.io simply.

R2: A list of orders will need to be generated to test the validation against, but this test data can be taken from this year's REST server though it will need to be adapted as the formatting is different to last years which the project was designed for so this will take some effort.

R3: Similarly to R2 data can be taken from this year's REST server but again some effort will be needed to fit it to the project

R4: The data for R2 and R3 can be re-used here, and then a manual inspection of having a timer report the time taken for completion for the tester to see will be sufficient.

4-Process and Risk

Due to the project already having been completed in the previous academic year (2023-2024) the testing timeline does not have to account for the simultaneous development of the project and so it can be broken down into this estimation:

Task	Start Date	End Date	Dependencies
Unit Testing	Day 1	Day 3	None
Integration Testing	Day 4	Day 5	Unit Testing
System Testing	Day 6	Day 8	Integration Testing
Performance Testing	Day 9	Day 10	System Testing
Compliance Testing	Day 11	Day 12	System Testing

There are a few potential risks associated with this testing process:

Errors in A* algorithm produce correct but inefficient solutions: If the implementation of the A* algorithm is even slightly wrong it could produce solutions that satisfy the requirements but are not optimal and this is a serious concern, but this can be checked manually by the tested with geoJson.io to visually check a range of paths.

Unrepresentative Test Data: The mock data used for testing might not accurately represent the actual data that the project will have to interact with. This is unlikely as the general structure and size of the data is known due to the REST server, however if this does occur it could impact all the requirement, so care is needed to ensure this does not occur.

Removal of the REST server: the REST server for the year that this project was created has been removed and thus there may be unseen complications with attempting to use this years new REST server as they may be changed to specification that are unaccounted for and so close care must be taken in using this as test data.