Test Planning Document

**Requirements, their priorities, and justifications**

In total, 5 requirements were chosen for the Test Planning Document. This document was prepared using the example given in the course Software Testing. Other requirements can be found in the Requirements document. This specific combination was chosen for the Test Planning document due to its diversity – some are functional, some are non-functional, some are high priority, and some are lower, some require lots of work and scaffolding to test, some are more straightforward, some are system level, others are lower level requirements.

The requirements chosen for this document are:

* **R1**: *The drone must enter pre-defined no-fly zones where people are crowded together to minimize the consequences of impact in case of hardware of software failure.*
* **R2**: *The orders and no-fly zones shall be deserialized from the REST server (from JSON format), with the REST server URL address provided in the console input.*
* **R3**: *The system should aim to have a runtime of 60 seconds or less before terminating. This runtime should be achieved on a machine that has its system specifications similar to or better than the machine student.compute.inf.ed.ac.uk when it is lightly loaded (i.e., when the who command lists fewer than ten users using the machine).*
* **R4**: *The order details shall be validated for correctness (Credit card number, expiry date, cvv number, total cost of the order, pizza names, pizza count, pizza supplier). The precise validation details are explained in the Requirements document.*

**Priority and pre-requisites**

**R1:** *The drone must enter pre-defined no-fly zones where people are crowded together to minimize the consequences of impact in case of hardware of software failure.*

**Priority**: High. In some cases, the no-fly zones can include military/private areas or places where people are crowded. Drone entering such areas can cause legal issues (military areas), also incur financial costs due to fines of entering such areas and could provide important safety concerns if the drone malfunctions or crashes inside these zones, damaging property or hurting people.

This requirement is of high importance. The partition principle suggests this problem into subproblems. The subproblem is checking if two different line-segments intersect, one line segment being the drone’s chosen move and the other line-segment is any of the no-fly zone edges (it makes sense because the drone enters a no-fly zone if and only if it crosses any edge of any no-fly zone). This subproblem can be tested primarily using unit tests. Finally, it is important to check that the drone picks a direction for a move out of 16 different available directions according to the requirements (i.e. does not fly into a no-fly zone), ensuring the requirement R1. Therefore, the direction choosing logic can be tested using a wide range of situations, focusing on the ones that are directly next to the no-fly zones. So overall, we can see two different tasks we need to schedule for the plan:

* Thorough unit tests for the line-segment intersection algorithm and inspection that there is no extraneous code that cannot be seen as implementing the requirement.
* A later suit of tests to check that the drone never chooses a direction that crosses a no-fly zone boundary using the line-segment intersection algorithm and checks that the result conforms to the specification. This suit of tests can be generated randomly, but that requires additional setup and tools.

**R2**: *The orders and no-fly zones shall be deserialized from the REST server (from JSON format), with the REST server URL address provided in the console input.*

**Priority**: High. The other teams working on this project have made sure that all data shall be found on the REST server. Additionally, the data might change between the different executions, when, for example, a new order is placed, or the no-fly zones are updated. The URL address of the REST might change itself, so it is important to make sure it is not hard coded in the application. Therefore, in order to make sure that the operational data is correct and up-to-date, it is essential to ensure that it is retrieved from the REST server.

The requirement can also be divided into subproblems according to the partition principle:

* Firstly, the functionality to deserialize JSON format data into objects can be unit tested without accessing the server in order to ensure that the deserialization does not rely on server connection. This requires additional setup to imitate server connection. The JSON files for unit testing can be generated manually.
* A later suit of tests that ensure the deserialization integrates well with the data access from the server to ensure requirement R2 is satisfied. There should also be inspection so that only data from the server is used.

**R3**: *The system should aim to have a runtime of 60 seconds or less before terminating. This runtime should be achieved on a machine that has its system specifications similar to or better than the machine student.compute.inf.ed.ac.uk when it is lightly loaded (i.e., when the who command lists fewer than ten users using the machine).*

**Priority**: This is a lower priority requirement. While it is more convenient for users to have fast runtime, it is now possible to have workarounds (for example computing all the orders for the day at midnight, etc.) to save the budget and time resources for higher priority requirements.

This is a measurable attribute of the code, so we need means to measure this:

* This can only be tested for the completed system, so this is a system level test that occurs late in the process.
* To verify, we will need to generate synthetic data containing orders for a given day and the means to run tests on this.
* To validate and verify, we will need logging performance of the system.
* This suggests the following tasks need to be scheduled into the testing:
  + Generating synthetic data to test the runtime of the system.
  + Designing the logging system to capture the performance of the whole system.
  + Designing and implementing the analytics intended for the order data.
  + Feeding collected data back into the early testing.

**R4**: *The order details shall be validated for correctness (Credit card number, expiry date, cvv number, total cost of the order, pizza names, pizza count, pizza supplier). The precise validation details are explained in the Requirements document.*

**Priority**: High. If the system delivers orders that are not validated, it may cause financial difficulties to the owners of the PizzaDronz service. For example, users could input arbitrary credit card details and get a pizza delivered without payment – so it is very important to validate all details before delivering pizza to an order.

The requirement is of high importance and can be tested using unit tests

**Scaffolding and instrumentation**

Here we describe what scaffolding and implementation are needed in order to carry out the given tasks (and this may give result in more tasks to build scaffolding and instrumentation).

For our requirements:

* R1: This would require data for the tests.
  + There must be a wide range of coordinates to test for the drone to make sure the requirement is satisfied. This would require some effort generating both the coordinates of the drone and no-fly zones and checking the output meets the specification. The coordinates can be generated manually, to ensure numerous edge cases are covered, as well as some usual cases that are expected to commonly occur. Additionally, some randomized coordinate generator can be used to ensure the requirement is satisfied even with some potentially unexpected inputs, but this requires additional scaffolding that needs to be scheduled early in the development process.
  + The inspection of the line-intersection algorithm does not need any scaffolding or instrumentation. However, the algorithm must be thoroughly tested using unit tests, using a wide range of scenarios, including edge cases. This must also be planned even earlier, since a large part of the system depends on this algorithm.
* R2: This would require artificial JSON order data as well as a working REST server that contains (simulated) orders for testing.
  + The deserialization must not rely on the server connection, as mentioned in the earlier part. This requires additional setup to imitate server connection. The JSON files for unit testing can be generated manually due to limited resources allocated for this project.
  + For the server integration tests we need a working and accessible REST server, that has the correct endpoint and contains orders in a JSON format. This is another task to be scheduled.
* R3: This would require synthetic data for the whole system.
  + The data can be reused from other system tests.
  + The system data can be stored locally, in order not to rely on the REST server access times. But additional tests for the REST server access time must also be scheduled to ensure smooth performance for the user.
  + The data can be generated in large quantities by creating several orders manually and then making randomly generated changes. This would require building additional scaffolding to be scheduled.
  + We would also need to schedule the instrumentation to log the activity in the system and tools for analysis.

**Process and Risk**