Planning document

Requirements

For this planning document I have chosen 2 requirements which are:

1. R1: The system should be able to check whether or not a given coordinate is within a given central area that is defined by other coordinates.
2. R2: The system should not allow the drone to fly in no-fly zones.

For the sake of variety, I chose R1 to be a unit level requirement and R2 – a system level requirement (these are the same requirements that were mentioned in 1.2).

Priority and pre-requisites

This section assesses each of the 2 requirements and also their T&A needs.

R1:

* This is a correctness requirement tested at unit level, so it will have a low priority
* Since tests can be performed at the unit level, testing should be done in early development stages, but can be done at any stage.
* Validation and verification checks will need to be performed.
* To verify and validate we will need synthetic test data where we already know the outputs, comprised of coordinates of a polygon and coordinates of a single point.
* The following inputs and outputs need to be considered as part of this requirement:

Inputs:

CentralArea: a set of coordinates that define the central area of Edinburgh city.

Point: a point that is a tuple of longitude and latitude.

Outputs:

IsWithinCentralArea: returns false if the point is not within the given polygon, otherwise true.

* Judging by how simple and low-level the requirement is, the only task that needs to be scheduled into the testing is generating synthetic data to test the method.

R2:

* This requirement has a far-reaching impact on multiple parties involved (security and privacy), therefore fulfilling it important, which gives it high priority
* High priority also means that the resources necessary to test will be high and that it would be best if the testing was done throughout the whole development phase to phase out as many bugs as early as possible before the system gets to build on them.
* At least 2 A&T principles will need to be considered because of the importance of this requirement, which will be redundancy and partitioning.
* We consider redundancy because redundant checks can increase the capabilities of catching specific faults early or more efficiently. For example, static type checking and validation of requirement specifications can reveal many type mismatches earlier and more efficiently during development. Additionally, assertion tests in code are redundant, but help increase confidence together with other tests.
* Partitioning (divide and conquer) also helps here because the requirement can be split into a functional component and a structural component. The functional component ensures that the drone never crosses or stops at a no-fly zone when going from one point to the other, whereas the structural component makes sure that does not happen for the entire day. Thus, the plan must include:
  + Some sort of examination of the code that checks if the algorithm ever sends the drone into a no-fly area.
  + A check after the system is built where we use many different order and available restaurant inputs too see if there is ever a fail of the specification.
* This requirement should take into account the following inputs and outputs:
  + Inputs:
    - noFlyZones: the list of no-fly zones for the central area.
    - currentDronePosition: the coordinates where the drone is located at the current tick
    - nextDronePosition: the coordinates where the drone will be located at the next tick
    - centralArea: the coordinates that define the central area of Edinburgh city
    - ordersCurrentDay: a list of orders to be processed for some specific day
    - ordersAll: a list of all orders for all days
    - restaurants: a list of participating restaurants
  + Outputs:
    - isDroneInNoFlyZone: if this is true then the drone has either crossed or stayed in the no-fly zone for at least one tick. If it’s false, then the drone did not cross a no-fly zone.
  + Specification:
    - isDroneInNoFlyZone is only true when a ray casting algorithm determines that a longitude and latitude tuple is within or crosses the boundaries of a polygon, which is a set of coordinates.
* The tasks that should be in the test plan after considering the two principles are generating synthetic test data, which will be used to test the drone flight algorithm, together with building some type of scaffolding to be able to run the software for the test. Also, system-wide synthetic data will be need to be generated and used to test the product at the highest level.

Scaffolding and instrumentation

This section outlines the necessary scaffolding and implementation for executing the given tasks.

R1:

* Scaffolding won't be necessary for this section as a straightforward unit test suffices to validate the requirement. As for instrumentation, only a scheduled unit test needs to be implemented.

R2:

* Some sort of simulator for the system, so that it’s possible to test the software. This is scaffolding and will need to be scheduled early.
* Having data for the simulator may involve some effort that needs to be scheduled
* The system test will use the simulator with the data obtained from the remote server and the lower-level tests will use the synthetically generated data instead

Process and Risk

R1:

The task of generating synthetic data and doing the testing should take no longer than 1 day. For this task it is necessary to have synthetic data to do the testing with. The task is also not very resource-intensive and is low-level, which should make it fairly easy to test. It could be placed in the design and implementation stage of the SRET lifecycle process.

The one risk involved with this task is that the synthetic data might be unrepresentative and so might not cover all the possible cases, which could make the design of the program flawed. One way to solve this problem is to make sure that the synthetic data has every possible combination to it so that no case is left untested.

R2:

A highly experienced programmer could probably complete the tasks for this requirement in 2 days, but for a more average programmer it might take anywhere between 3 to 6 days. To be able to test properly, data from the remote server will be needed (which includes restaurants, central area coordinates, no-fly zones). The requirement falls under the design and implementation stage of the SRET lifecycle process due to it being a major factor which must be complied with for safety purposes.

Potential risks include the drone flying over no-fly zones, which could lead the drone to crash if it happens to hit a tall structure, and also public safety and privacy, where the civilians could get annoyed of the buzzing sound or fell like their private space has been invaded. To mitigate this risk, adequate testing would need to be done throughout the entire development process. There is also risk of taking unexpectedly long to complete the requirement, depending on how complicated and accurate the path finding algorithm is supposed to be. This can be tackled by putting a higher number of resources into completing the requirement.

Evaluation of the instrumentation

The existing instrumentation combines synthetic data generated at the unit level with real data for high-level testing, resulting in comprehensive testing. These tests ensure adequate coverage, providing assurance that the system operates as intended or enabling the discovery and correction of any necessary bugs in the worst case. However, in the case that the synthetic data cannot cover all the possible test cases, there is a chance of a fault staying within the system after not being detected, possibly leading to something catastrophic.

Manual testing has the potential to enhance the robustness of our system, but it has its own limitations. The developer, who is aware of the system's inner workings, would most likely carry out the manual testing, leading to a lack of thorough testing as the developer may not attempt to make certain inputs that an untrained user might make. Furthermore, manual testing is time-consuming and resource-intensive, making it a challenge within the constraints of a tight schedule.