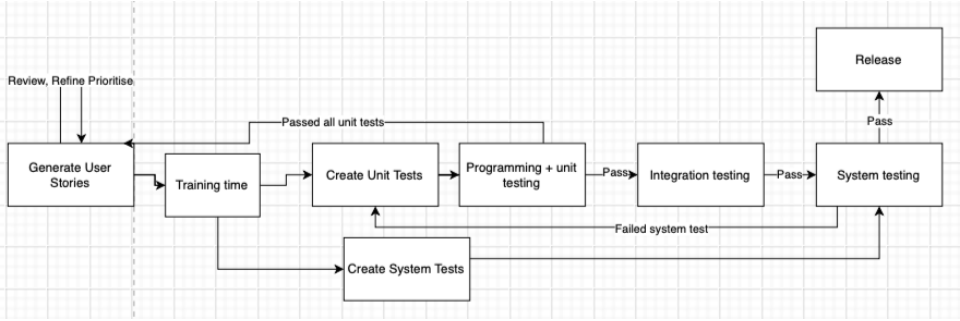
Test-Planning Document

Chosen Lifecycle

My chosen lifecycle is an adaptation of Extreme Programming:



In my adaptation, acceptance testing is replaced with system testing as no one from the organisation requesting the software can do any testing, so I must perform it myself. So too pair programming is replaced with programming as the project must be a solo project so no-one can review my code but me. I have also added a stage for integration testing as I would like to separately test this in order to ensure the quality of my software.

Priority and Prerequisites

R1. Drone does not pass through any no-fly zones.

Discussion

This is a safety requirement, various faculties within the UoE have expressed safety concerns and so violating this requirement could lead to termination of the project or regulatory issues. As the drone will only fly in straight lines we can avoid ever entering no-fly zones by checking whether a proposed flight enters them, if it does, we can simply choose another route and check again.

- As it is critical to the safety and viability of the drone this requirement should be devoted a high level of resource.
- The principle of timeliness (chapter 4) indicates that we should aim to detect errors as soon as possible in the development process, to this end we should test this functionality independently as soon as the code is developed rather than waiting for integration with other areas of code (this may require scaffolding).
- As it has high priority, we should consider at least two T&A approaches in accordance with chapter 3.
- In line with the principle of sensitivity (chapter 3) we should aim to increase the sensitivity of this function so that if it fails for one test case it will fail for all. This would in turn increase the visibility of any faults in the code.

Fits into lifecycle:

- This requirement should be tested at the “programming + unit testing”, “integration testing” and at the “system” phase. This is because this is a safety requirement and therefore it is imperative that it is tested thoroughly. There are various areas where this functionality could go wrong: at unit level when the function used to decide if a path enters a no-fly zone, at the integration stage when deciding whether to take a route based on the unit just mentioned, or at the system level when obtaining the no fly zones, storing them, forming routes etc. Therefore, we must test at all three stages.

Input, Output, Spec

The inputs and outputs for ‘possible()’ function are as follows. This is the unit that will be tested as an example, but similar input/output/spec should be produced for the integration and system level testing will be done

Inputs:

- Starting longlat: longlat object containing longitude and latitude of starting coordinate.
- End Longlat: longlat object containing longitude and latitude for destination coordinate.
- noGos: an array of polygons marking the no-fly zones.

Output:

- Possible: True if proposed route doesn’t cross through a no-fly zone, False if it does.

Specification:

- Function 'possible()' returns true if the route between two coordinates doesn't cross any no-fly zones and returns false if it does.

R2. Given two coordinates the system can generate a series of moves moving the drone from start to 'close to' destination, avoiding no-fly zones and staying within the confinement area, making a record of each move it makes in order to reach destination.

Discussion

This is a core piece of functionality that ties together various parts in order to move between two coordinates. As it is central to the system working it is of high importance, however the actual functionality isn’t too complicated, and it is more about tying together various other pieces of code.

- This suggests a medium to high level of resource as it is an important functionality but not overly complicated.
- This functionality comes quite late in development after other components that contribute to it are finished so this is an integration test that occurs later in our schedule.
- We will need synthetic data (simply coordinates to go between)
- Other components needed for this should be scheduled to be tested first to ensure that they are robust before this functionality is implemented.

- I would like to schedule it to be tested once other 'movement' functions have been implemented but before it is integrated with the functions that obtain no-fly zones and features. Therefore, scaffolding will be required to simulate these areas.

Fits into lifecycle:

Testing this piece of functionality should take place in the "integration testing" phase of testing. This is because it is about tying together various units of functionality. These areas of functionality are covered by other requirements and therefore will have already been tested in the unit testing phase.

Input, Output, Spec

The inputs and outputs for moveTo() are as follows:

Inputs:

- Starting longlat: longlat object containing longitude and latitude of starting coordinate.
- End longlat: longlat object containing longitude and latitude for destination coordinate.

Output:

- Array of Points: an array containing coordinates of each point visited in chronological order to move from one point to another

Specification:

- MoveTo() can take two points as input and output the points visited that constitute a route between the points that atones to all restrictions on flight.

R3. System should run in reasonable time to ensure viability (mean <60s)

Discussion

This is important in order to ensure the viability of the system, however unacceptable times of 10+minutes are far off the goal of a mean of <60 seconds. Therefore, while care should be taken to ensure all components aren't unnecessarily slow achieving <60 seconds should be seen as less important than the system achieving full functionality. This is a measurable attribute of the system, and we will therefore need the means to measure this.

- This suggests a medium to low level of importance, resources should be spent ensuring that the system is as fast as possible within reason, but this should not come at the cost of functionality, which is more important.
- This is a system test and so should occur late in our schedule, however, we should test large sections of functionality early and throughout the schedule such as 'movement', 'orders' and 'restaurants' in order to ensure that we don't end up with one section hugely slowing down the full system.
- To verify this requirement, we will need synthetic data to run the system repeatedly and obtain a mean runtime.
- We will have to log the runtime in order to validate this.

- We will therefore have to schedule the following tasks:
    - Generate synthetic data for a typical day's orders, so we can run the full system.
    - Scaffolding in order to test individual sections of functionality for reasonable runtime
    - A means to evaluate which sections run for how long when we test the full system.

Fits into lifecycle

This requirement should be tested at “system testing” phase. This is because it is testing the software in its final form. However, upon completion of each module (e.g., movement, orders etc.) the module should be briefly tested, and its runtime noted, in order to catch any glaringly large runtimes early before we perform more integration in line with the principle of timeliness (chapter 4). So limited testing at integration phase with more comprehensive testing performed at system level.

Input, Output, Spec

Input:

- Day; integer
- Month; integer
- Year; integer
- Web port; integer, port which web server will be run on
- Database port; integer, port which database will be run on

Output:

- Time; integer, time in seconds

Specification

- The system must successfully perform all core functions for the time to be considered this is limited to; produces viable flightpath, creates database containing orders delivered for day, creates, creates a list of coordinates representing the moves made in delivering the orders for a day.

R4 Drone can only fly in a direction that is a multiple of 10 from 0 to 350

Discussion

This requirement is mentioned in the specification provided by the informatics department however it isn’t vital to the operation of the system (the system would still operate if any angle of travel was allowed), it is probably added to simplify the flightpath. It also isn’t a safety issue. The functionality is simple and shouldn’t be overly complicated to implement.

- Due to its simplicity and being relatively inconsequential to the operation of the system this requirement should be given a low level of importance.
- This is a unit test and therefore should take place early in development. The test should be created in the ‘create unit tests’ phase and applied in the ‘programming + unit testing’ phase of testing.

- To implement this test, we will need synthetic data acting as the input for current position and angle of flight
- We will therefore have to schedule the following events:
    - Generate synthetic data
    - Create unit test
    - Run unit test

Fits into lifecycle:

This will fit into the 'create unit tests' and 'programming + unit testing' phases of the chosen lifecycle. This is because the function shouldn't rely on any other sections of code and therefore following the principle of timeliness, we should aim to detect any potential errors as soon as possible, therefore we should test this functionality as early as possible which is in the aforementioned phases.

Input, Output, Spec

Inputs: A varying range of:

- Startpoints – longlat objects representing the starting coordinates.
- Angles – int values dictating which direction we wish to make a move in.

Output:

- Direction – an int value for the direction, within the specified range

Specification:

- The function should always return an angle which is a multiple of 10 and between 0 and 350 (inclusive) so long as the input is acceptable, otherwise it should return an error.

R5 Drone can test whether a coordinate is 'close to' (within 0.00015 degrees) another coordinates

Discussion

This requirement is important in that it affects lots of later areas of functionality and allows us to get close to restaurants/customers to pick up/drop off orders. However, the functionality should be simple to implement. This suggests a medium to low level of importance. This function relies on the ability to calculate the distance between two coordinates, this functionality may be implemented in another function or in the same function. It is a unit test and should be tested early as it doesn't rely on much/any other functionality.

- Function should be given medium – low importance as it is simple to implement but reasonably important to the functioning of the system.
- This is a unit test and therefore should take place early in development
- We will need some synthetic data in order to test whether two coordinates are close to each other, we will need examples of coordinates that are close and ones that aren't as well as some bogus input.

Fits into lifecycle:

This testing should take place in the 'create unit tests' and 'programming + unit testing' phases of the lifecycle, this is because the function is simple and relies on minimal other functionality and therefore can and should be tested as early as possible.

Input, Output, Spec

Inputs:

- Coordinate1: a longlat object representing the first coordinate
- Coordinate2: a longlat object representing the second coordinate

Output:

- CloseTo: a Boolean returning True if the two coordinates are close to each other and false otherwise

Specification:

- The function should return a Boolean value; True if the two coordinates are close to each other and false otherwise.

Scaffolding and Instrumentation

For our requirements the scaffolding and implementation that is needed is:

R1: The testing will require quite extensive scaffolding; this functionality relies on various other aspects of the system working and so scaffolding will be needed to simulate the running of these parts. Synthetic data will also need to be supplied.

- Specifically, we may not yet be able to rely on software that reads the list of no-fly zones from the webserver. Therefore, we will have to simulate data on the no-fly zones.
- Later we will have to test that this function integrates with the function that obtains and manipulates the no-fly zones.
- We will have to simulate potential paths in order to test whether the function can decide if they are possible or not.

R2: This testing will require extensive scaffolding. As this function ties together other areas of code, it is reliant on closeTo, angle, distanceTo functions as well as reading from the webserver and many other areas too. For this reason, scaffolding will be needed to either simulate or run the other areas of functionality.

- Scaffolding to simulate data for no-fly zones and start and end coordinates.
- Scaffolding to simulate other areas of code depending on the development stage.

R3: This is a system level requirement and will require simulated data, for example days order, menus, restaurants features etc.

- In order to test the full system's runtime, we will need to provide synthetic data to run the whole system.
- Will need to provide scaffolding to run each section independently to obtain an idea of which sections are slowest.

R4: this will require limited scaffolding; simply to create an instance of the current location, and then to call the function with a given angle.

R5: this will require almost no scaffolding at all, simply calling the closeTo function with two longlat coordinates as input.