Testing Techniques

Functional Testing

Brute force approach:

In order to exercise the use of brute force testing I will use it to test the following requirement:

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

I will form the test suite by looking at the requirement and simply developing the test cases myself just form the requirement.

<u>Test specification:</u>

Given two longlat coordinates, the closeTo() function returns true if they are within 0.00015 degrees and false otherwise.

<u>Test case:</u>

<u>Inputs:</u>

1. (0.0, 0.0), (0.0, 0.0) 2. (0.0, 0.0) (0.0, 0.00015) 3. (0.0, 0.0) (0.00015, 0.0) 4. (0.0, 0.0) (0.0, 0.00014) 5. (0.0, 0.0) (0.00014, 0.0) 6. (0.0, 0.0) (0.00007, 0.00007) 7. (0.0, 0.0) (12, 12) 8. (-3.186874, 55.944494), (-3.186767933982822, 55.94460006601717) 9. (-3.186874, 55.944494), (-3.1873,55.9430) 10. (-1, -1), (-1, -1.00014) 11. (0,0)

<u>Expected Outputs:</u>

1. True 2. False 3. False 4. True 5. True 6. True 7. False 8. True 9. False 10. True 11. Error

<u>Execution conditions:</u>

The first longlat must be created as a longlat class and the second longlat should be supplied as the input to the closeTo function of that class.

Pass/fail criterion:

Pass – output of the closeTo function is as specified in expected outputs

Fail – the output is anything else

Example test:

```
Eoin Reid
@Test
public void testCloseToTrue(){
    LongLat alsoAppletonTower = new LongLat( longi: -3.186767933982822,  lat: 55.94460006601717);
    assertTrue(appletonTower.closeTo(alsoAppletonTower));
}
```

Random approach:

In order to exercise the use of random testing I will use it to test the following requirement:

The system can calculate the delivery cost for a given list of items (unit)

I will form the input by randomly generating combinations of food items in order to test the function that calculates the cost of a given list.

Test case specification

Given a list of between 1 and 5 items, the getDeliveryCost function returns an integer value representing the delivery cost.

Execute conditions

The webserver must be running, and a menus class must be instantiated before the test is run.

Test case:

Randomly generated inputs:

1. { "item" : "Flaming tiger latte", "pence" : 460 },

{ "item" : "Dirty matcha latte", "pence" : 460 },

{ "item" : "Strawberry matcha latte", "pence" : 460 },

{ "item" : "Fresh taro latte", "pence" : 460 },

2. { "item" : "Guava fruit tea", "pence" : 380 },

{ "item" : "Apple fruit tea", "pence" : 380 },

{ "item" : "Strawberry fruit tea", "pence" : 380 },

{ "item" : "Lemon fruit tea", "pence" : 380 },

3. { "item" : "Coronation chicken sandwich (Large)", "pence" : 400 },

{ "item" : "Bacon, brie and cranberry sandwich (Regular)", "pence" : 320 },

{ "item" : "Bacon, brie and cranberry sandwich (Large)", "pence" : 400 },

etc.

Expected output

1. 1890
2. 1570
3. 1170

Pass fail criterion

For each random input the test passes if the output matches the expected output and fails otherwise.

Example Test:

```java
Eoin Reid +1 *
@Test
public void testMenusFourA() {
    // The webserver must be running on port 9898 to run this test.
    Menus menus = new Menus( port: "8080");
    ArrayList<Menu> menusList = menus.getMenus();
    ArrayList<String> items = new ArrayList<>(Arrays.asList(
            "Ham and mozzarella Italian roll",
            "Salami and Swiss Italian roll",
            "Flaming tiger latte",
            "Dirty matcha latte"
    ));
    int totalCost = menus.getDeliveryCost(items);
    // Don't forget the standard delivery charge of 50p
    assertEquals( expected: 230 + 230 + 460 + 460 + 50, totalCost);
}
```

Systematic approach:

Requirement I'll test using this technique:

Given two coordinates the system can generate a series of moves moving the drone from one to another, avoiding no-fly zones and staying within the confinement area.

1. First step is to identify independently testable features (ITFs):
- System can test whether a coordinate is in the confinement area
    - Inputs – coordinate, confinement area (static value)
- System can test whether a proposed route enters any no-fly zones
    - Inputs – current and next, noFlyzones
- System can make a record of a move it has made
    - Inputs – current, next, angle, orderNo
- System can tell when it is close to its destination coordinate
    - Inputs – coord1, coord2
- System can generate series of moves from A to B avoiding no fly zones and staying within confinement area
    - All above mentioned
2. Partition categories and constraints
    a. System can test whether a coordinate is in the confinement area
        i. Coordinate – empty, within confinement area, on edge of confinement area, outside confinement area
        ii. Confinement area – empty, nonempty
    b. System can test whether a proposed route enters any no-fly zones

        i. StartCoord, EndCoord – empty, path doesn't cross no-fly zones, path does cross no-fly zones
        ii. NoFlyZones – empty, nonempty
    c. System can make a record of a move it has made
        i. Current – empty, nonempty
        ii. Next – empty, nonempty
        iii. angle – empty, nonempty
        iv. OrderNo – empty, nonempty
    d. System can tell when it is close to destination coordinate
        i. Current – empty, is close to, isn't close to
        ii. Destination – empty, is close to, is close to.
3. Write and process test specifications
    a.
        i. Consider case where coordinate is empty
        ii. Consider case where coordinate is within confinement area
        iii. Consider case where coordinate is on edge of confinement area
        iv. Consider case where coordinate is outside of confinement area
    b.
        i. Consider case where start and end coords are empty
        ii. Consider case where path crosses no-fly zones
        iii. Consider case where path doesn't cross any no-fly zones
        iv. Consider case where no-fly zones is empty
    c.
        i. Consider cases where each of the inputs is empty
        ii. Consider cases where one (test for each) variable is empty
        iii. Consider case where all are nonempty
    d.
        i. Consider case where both coords are empty
        ii. Consider case where neither coords are empty and:
            1. Coords are close to each other
            2. Coords aren't close to each other
4. Create test cases
    a. .
        i. <u>Input</u>

() - current, () - next () - No-fly zones

<u>Expected output</u>

False

<u>Pass fail criterion</u>

The test should pass if False is returned and fail if true is returned

        ii. <u>Input</u>

(-3.186874, 55.944494) - within (|55.942617, 55.946233, -3.192473, -3.184319)

Expected output

True

Execution conditions

The input must be saved as the current coordinates in the longlat object before running

Pass fail criterion

The test should pass if the function returns True and fail otherwise

iii. Input

(|55.942617,, -3.184319) - edge (|55.942617, 55.946233, -3.192473, -3.184319)

Expected output

True

Execution conditions

The input must be saved as the current coordinates in the longlat object before running

Pass fail criterion

The test should pass if the function returns True and fail otherwise

iv. Input

(-3.1928,55.9469) - outside (|55.942617, 55.946233, -3.192473, -3.184319)

Expected output

False

Execution conditions

The input must be saved as the current coordinates in the longlat object before running

Pass fail criterion

The test should pass if the function returns False and fail otherwise

b.

i. Input

() - current, () - next, No-fly zones downloaded from webserver

Expected output

True

Pass fail criterion

The test should pass if True is returned and fail otherwise

ii. Input

(55.945626,-3.191065) - start, (*-3.186874, 55.944494*) - next, nofly zones downloaded from webserver

Expected output

False

Execution conditions

The input must be saved as the current coordinates in the longlat object before running

Pass fail criterion

The test should pass if the function returns False and fail otherwise

iii. Input

(55.945868,-3.188656)- start, (*-3.186874, 55.944494*) - next, nofly zones downloaded from web server

Expected output

True

Execution conditions

The input must be saved as the current coordinates in the longlat object before running

Pass fail criterion

The test should pass if the function returns True and fail otherwise

Due to time and resource constraints, I am only including the test cases for the first two ITFs, however this shows that I understand the process and would continue to do the same for the others.

Example tests:

IsConfined:

```
Eoin Reid
@Test
public void testIsConfinedTrueA() { assertTrue(appletonTower.isConfined()); }
```

IsPossible:

```java
eoinpatrickreid *
@Test
public void testPossible() {
    // The database must be running on port 9751 to run this test.
    Movement movement = new Movement( p: "8080");
    movement.getFeatures();
    movement.getLandmarks();
    //movement.getEdges();
    OrderDetails orderDetails = new OrderDetails( dport: "9751");
    Menus menus = new Menus( port: "8080");
    ArrayList<Menu> menusList = menus.getMenus();
   Date date = new GregorianCalendar( year: 2023, Calendar.DECEMBER,  dayOfMonth: 31).getTime();
   String jdbcString = String.format("jdbc:derby://localhost:%s/derbyDB", "9751");
   Orders orders = new Orders(jdbcString, date, orderDetails, menus);
   ArrayList<Order> ordersList = new ArrayList<Order>();
   ordersList = orders.getOrders();
   ordersList = (ArrayList<Order>) ordersList.stream().sorted(Comparator.comparing(Order::getValue).reversed()).collect((Collectors.toList()));
    Words words = new Words( p: "8080");
    LongLat x = words.locationFromWords( input: "sketch.spill.puzzle");
    LongLat y = new LongLat( longi: -3.186874,  lat: 55.944494);
    //ArrayList<String> detailsList = orderDetails.getOrderDetails("62b4f805");
    assertFalse(movement.possible(x, y));
}
```

Combinatorial Testing

Catalogue based testing:

Requirement I'll test using this technique:

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

Categories:

1. The element immediately preceding the lower bound of the interval

2. The lower bound of the interval

3. A non-boundary element within the interval

4. The upper bound of the interval

5. The element immediately following the upper bound

<u>Test case specification:</u>

The angle between a given longlat and the result of nextPosition(int x) (for a range of x) should be between 0 and 350 and divisible by 10 or an error if the input is incorrect.

<u>Test case:</u>

Inputs based on categories:

```
-1 0 1 2 15 17 57 60 67 73 77 88 89 95 108 111 122 126 130 137 142 144 148
151 157 158 166 193 197 198 208 217 219 226 229 235 243 247 264 265 273
283 284 289 317 346 349 357 359 360 361
```

<u>Execution conditions:</u>

A longlat must be instantiated in order to run the test, choose (0.0, 0.0) to be the longlat coordinates

<u>Pass fail criterion:</u>

Pass – angle between result of nextPosition() and the longlat we provided is 0<=angle<=350 and is divisible by 10. Or error is thrown if input is incorrect

Fail – angle is anything else, error isn't thrown on incorrect input.

Example Test:

```
public void testAngle190(){
    LongLat nextPosition = appletonTower.nextPosition( angle: 190);
    LongLat calculatedPosition = new LongLat( longi: -3.1870217211629517,  lat: 55.94446795277335);
    assertTrue(approxEq(nextPosition, calculatedPosition));
}
```