Ecolocation

Software Testing Plan

Version 1.0

April 4, 2018

Client/Mentor:
Dr. Chris Doughty

Team Members:
Brenden Bernal
Chandler Hayes
Michael Hartzell
Anthony De La Torre

Table of Contents

Introduction

The purpose of this document is to discuss the different components of our software testing plan for our Android mobile app. The application is an educational tool for the general public to emphasize how animals play an indispensable role in providing for the overall health of our ecosystems. It does this by displaying location-based ecological information in an easily understandable fashion. To provide ease of access, the app removes the tasks of exploring different sources for information, it removes the necessity of using MATLAB, and it removes the need for an understanding in ecology. This allows users to explore any ecosystem on land and get information about the mammals that live in it, and how they affect it. Additionally, it provides further knowledge of how ecosystems have changed over thousands of years by providing the ecosystem's state during the Pleistocene Era. By using this app, we want users to become more educated in the important role animals play and to become more motivated in learning more about the environments.

To ensure a great user experience, the app will be tested in three main areas: unit testing, integration testing, and usability testing. The unit testing section covers how we are testing the individual parts of modules in the app. This section will primarily focus on the methods related to calculating the nutrient movement of a specific ecosystem and methods for retrieving the list of mammals in a specific location. The integration testing section describes how we will test how different modules of the app communicate with each other. This will include how the app communicates with the database, retrieves datimages, and how different activities transfer important data amongst themselves. The usability testing will describe how we plan to test the user experience with test subjects using the app. By utilizing these three main testing methods, we should be able to find and fix any problems in the app.

Unit Testing

The unit testing for our application is not the most extensive because many of the methods in the app are for configuring the **user interface** (UI) components of the app to display data. For every activity and fragment within the app, the onCreate or onCreateView methods are called to set up the UI. The work done in these methods are retrieving attributes from objects and displaying through the use of widgets. Another component we will not be executing unit tests on are the methods related to spatial mapping. Many of the methods for spatial mapping are using static data and uses Google Maps' Heat Map Utility in its most basic form. Additionally, a lot of the code that was written for this application are meant to be used once acquire additional needed for the database and to configure the data needed into the appropriate format for the database.

The areas of the application that we will execute unit tests on are for sorting the list of mammals in the given ecosystem, calculating nutrient movements, methods in the Ecosystem class, and methods for getting coordinates from a text box. The methods related to these components are more dynamic and have situations in which errors can occur due to individual methods. The unit tests will be constructed using JUnit and Mockito.

Ecosystem Unit Tests

The Ecosystem class contains the list of current mammals, the list of historic mammals (from the Pleistocene Era), and the location the user selected. This class enables different activities to have all the necessary animal data and location data needed to perform necessary functions. This class is also responsible for retrieving the current and historic animals from the database.

getAnimal(String scientificName, AnimalType type)

This method retrieves a specific animal from the either the current list of mammals or the historic list of mammals depending on the value of AnimalType.

Inputs to Test	**Expected Outcome**
Animal that is in the current animal list	Returns animal from current animal list and has the same scientific name
Animal that is in the historic animal list	Returns animal from historic list and has the same scientific name
Scientific name that does not exist in either list	Return null object
Use animal type that does not match the wanted animal	Return null object

getCurrentData(final LatLng coordinates)

This method retrieves all the current mammals for the specified location from the database.

Inputs to Test	**Expected Outcome**
Land coordinates	Returns an ArrayList of animals that currently exist in given location
Ocean coordinates	Returns empty ArrayList
Non-existing coordinates (eg: (400,250))	Returns empty ArrayList

getHistoricData(final LatLng coordinates)

This method retrieves all the historic mammals for the specified location from the database.

Inputs to Test	**Expected Outcome**
Land coordinates	Returns an ArrayList of animals that existed in given location during Pleistocene Era
Ocean coordinates	Returns empty arraylist
Non-existing coordinates (eg: (-400,400))	Returns empty arraylist

loadImageFromURL(final Animal animal)

This method retrieves the image that corresponds to the given animal. It can handle cases where there is no corresponding image by setting a predetermined image when the image does not exist.

Inputs to Test	**Expected Outcome**
A current animal	Sets image for given animal
A historic animal	Sets image for given animal
Animal that does not have a picture stored	Sets generic image for given animal

loadRangeMapFromURL(final Animal animal)

This method retrieves the range map for the given animal and sets the range map to the given animal. This method is designed only for historic animals.

Inputs to Test	**Expected Outcome**
Historic animal	Sets its range map image to the animal
Current animal	Does not set a range map image to the animal

Nutrient Movement Unit Tests

nutrientMovementCurrent(ArrayList CurrentAnimals)

This method takes the list of current animals in the selected location and takes the mass from each animal in the list. It then adds up all the masses and adds it to the BarChart.

Input to Test	**Expected Outcome**
Current animal list where sum is > 0	Calculates the nutrient movement for the current mammals in the ecosystem should show nutrient movement greater than 0.
Current animal list where sum = 0	Calculates the nutrient movement for the current mammals in the ecosystem should show nutrient movement being 0.

nutrientMovementHistoric(ArrayList HistoricAnimals, ArrayList CurrentAnimals)

This method takes the list of historic animals in the selected location and takes the mass from each animal in the list. It then uses nutrientMovementCurrent() to get the current nutrient to add to the historic nutrient movement.

Input to Test	**Expected Outcome**
Historic animals list where sum is 0, Current animals list where sum is > 0	Historic nutrient movement will match current nutrient movement.
Historic animals list where sum > 0, Current animals list where sum is is 0	Historic nutrient movement will be greater than current nutrient movement.
Historic animals list where sum is 0, Current animals list where sum is 0	Historic nutrient movement will match current nutrient movement.
Historic animals list where sum is > 0 Current animals list where sum is > 0,	Historic nutrient movement will be greater than current nutrient movement.

What-If Scenario Unit Tests

whatIf(slider position 0, slider position 1)

What if uses a range slider listener to take inputs from the user for the range of weights to take in.

Input to Test	**Expected Outcome**
Slider positions (0,100)	Nutrient distribution for all mammals from 0kg to 100kg
Slider positions (50, 100)	Nutrient distribution for all mammals from 50kg to 100kg
Slider positions start with (50-100) and change to Slider position of (0, 100)	Nutrient distribution for all mammals from 0kg to 100kg
Slider positions start with (0-50) and change to Slider position of (0, 100)	Nutrient distribution for all mammals from 0kg to 100kg
Slider positions (0, 50)	Nutrient distribution for all mammals from 0kg to 50kg
Slider positions (25, 50)	Nutrient distribution for all mammals from 25kg to 50kg

Location Unit Tests

The Location activity is responsible for letting the user choose the location of the ecosystem they want to explore. It gives the user different ways of selecting a location: user's location (if permission is granted), the default location, a custom location. To enter a custom location, the user can either drag and drop the map marker or enter the coordinates for a custom location.

getDeviceLocation()

This tries to get the most recent location the phone's device has stored. It only gets the most recent location if permission to access the user's location is granted and the device has a recent location stored. The permission granted is stored in class variable called *isPermissionGranted*. This method also uses the class variable called *chosenLocation* which stores the currently selected location. Once the most recent location has been retrieved, the results are stored in variable called *locationResult*.

Inputs to Test	Expected Outcome
isPermissionGranted: True chosenLocation: (40, 40) locationResult: (70, 70)	Places marker at (70, 70)
isPermissionGranted: True chosenLocation: (40, 40) locationResult: null	Places marker at (40, 40)
isPermissionGranted: False chosenLocation: (40, 40) locationResult: null	Places marker at (40, 40)

updateTextViews()

Gets the values from the Latitude text box and the Longitude text box to get the inputted coordinates. Testing this method will require running tests that solely test the inputs for latitude and then solely test the inputs for longitude. By testing this way, we can ensure that inputs for latitude and longitude can work independently of each other.

This table represents inputs to test for latitude.

Inputs to Test for Latitude	Expected Outcome
“90”	Chosen location’s latitude will be (90, 0)
“-90”	Chosen location’s latitude will be (-90, 0)
“91”	Does not change chosen location and displays wrong input dialog
“-91”	Does not change chosen location and displays wrong input dialog
“word”	Does not change chosen location and displays wrong input dialog
“(0”	Chosen location’s latitude will be (0, 0)
null	Does not change chosen location and displays wrong input dialog

This table represents inputs to test for longitude and the latitude is 0°.

Inputs to Test	**Expected Outcome**
“180”	Chosen location’s latitude will be (0, 180)
“-180”	Chosen location’s latitude will be (0, -180)
“181”	Does not change chosen location and displays wrong input dialog
“-181”	Does not change chosen location and displays wrong input dialog
“0)”	Chosen location’s latitude will be (0, 0)
“word”	Does not change chosen location and displays wrong input dialog
null	Does not change chosen location and displays wrong input dialog

Sorting Unit Tests

There is one main method for sorting the data in our application. In order to test the efficacy of our sorting algorithms JUnit tests were created. These tests tested the methods with different potential inputs to check that our algorithms worked with any possible combination of data.

sort(ArrayList<Animal> list, SORT_TYPE sortType, int order)

This method can sort Animal objects by strings or numbers in ascending or descending order. Specifically, it sorts an ArrayList that contains Animal objects by its common name, scientific name, mass, or threat level. The threat level is represented as an enumeration where each enumeration holds an integer value; this enables sorting by numbers. Each test case will use the same instance of an ArrayList<Animal>, inputs that indicates which Animal attribute is being sorted and in what order (ascending or descending).

Case to Test	Expected Outcome
Common name, ascending	The list of animals is in alphabetical order by common name
Common name, descending	The list of animals is in reverse alphabetical order by common name
Scientific name, ascending	The list of animals is in alphabetical order by scientific name
Scientific name, descending	The list of animals is in reverse alphabetical order by scientific name
Mass, ascending	The list of animals is in ascending order by mass
Mass, descending	The list of animals is in descending order by mass
Threat level, ascending	The list of animals is sorted by least threatened to extinct
Threat level, descending	The list of animals is sorted by extinct to least threatened

Integration Testing

The following Integration tests will be performed on our application to ensure that all of our modules work together to share data accurately. The three main modules that we are testing in this section are the app itself, the database, and the private website that stores the animal and range map images. We are testing the flow of data between the database and the application, the flow of data between the photo URL and the application, and the flow of data within the app itself.

Database-App Integration Tests

For testing the flow of data between the application and the database we have two parts to test; how the app works with the PHP script and how the PHP script works with the database. For both of these tests we will be adding lines into the actual code to ensure the integrity of the data sent and received while the application is running. These are both simple and straightforward tests but they will validate that the data is flowing throughout the app successfully. By ensuring the integrity of the database integration

we can be confident that the results and the data presented in the application are accurate.

App and PHP Script

There are two tests for this integration. The first test is how the app shares data with the PHP script. The test here will be to make sure that the coordinate data is being sent from the app to the PHP script correctly. Our tests will utilize print statements on the PHP side to output the data that is received and we will be able to manually inspect the results to see if the the tests pass.

The second test is validating the return from the PHP script to the app. The app should receive a list of animals at the users input along with all the information on that animal. This test will consist of outputting this list and comparing the results against an SQL script. The SQL script will be run manually by inputting the coordinates and then the results will be compared to the list returned from the PHP script.

PHP Script and Database

There will be a single test for the integration of the PHP script and the database. This tests will simply compare the list returned to the PHP script against an SQL query with the same input. By manually comparing the two lists we can check to see if the proper data is being returned and that the PHP script is correctly interacting with the database.

Images-App Integration Tests

One of the most important element of the animal list is the image for each animal. In order to ensure the animals we have images for appear, we are running two tests. The first focuses on testing to make sure the URL for an animal actually returns an image. This will be done by manually entering the URL into a web browser. The second test focuses on making sure a validated image URL appears on the app using Picasso.

Picture URL to Image

In order to check the validity of our URLs, we will test a base set of animals from the database to retrieve the images for. Our plan is to take 10 animals from the current database and 10 animals from the historic database. The URL to retrieve the image for each animal will be tested, as well as the range map image for the historic animals. Several invalid URLs will also be used, to make sure no unexpected results occur.

To perform the test, the URLs will be manually entered into a web browser. For a valid URL, the image of the animal will appear. If it is invalid, the webpage should return an error. This will ensure that the URLs are retrieving the correct image.

Picasso to App

To check that Picasso is retrieving the correct images, the validated URLs from the previous test will be used, along with several invalid URLs, and several images not from our website. This will give us a better understanding of what the potential problems are and where they are located.

To test each animal, we will begin by determining one location that animal lives in. We will run the app for that location, and ensure that the correct image appears next to the given animal. Then the animal will be clicked on to move to the detailed information page, to check if there are problems displaying the image on that page. Historic animals will also have their range maps tested at this point. To test the other URLs we will replace the standard URL, with the test URL and do the same tests. Invalid URLs should display a generic image. By covering all of these situations, we should be able to find any potential problems with the images.

App-to-App Integration Tests

In order for the app to behave properly, many of the activities need access to the lists of current and historic animals for chosen location. The Ecosystem class was created to store all the information on the lists of animals for the chosen location and to store the coordinates of the chosen location. The following integration tests will verify that the different activities that needs this information can successfully communicate with the single instance of the Ecosystem class to get the needed information.

Animal Data to Activities

To test the reliability of retrieving data from the Ecosystem object, the list of current and historic animals will be retrieved from each activity that needs access to this data. This means that there will be three similar tests to run, which will be run consecutively. The first test will be for the ListView activity, the second test will be for the Detail Activity, and the final test will be for the GraphResults Activity. These tests will verify that the retrieval of the animal data is accurate and consistent. The tests will start by initializing the current and historic animal lists with at least 10 animals. Each test will require

retrieving and printing the lists retrieved. The test is successful if all of the lists from each of the three activities are identical. These tests will be re-run, when the chosen location changes to ensure that the list of animals updated appropriately.

Results to MPAndroidChart

To test the reliability of sending results to the BarChart, that is made using MPAndroidChart, we will be sending the relevant data to the BarChart. The relevant data would be the current animal nutrient movement, the historic animal movement, and the what-if scenario. The current and historic nutrient movement are only calculated once after the location is chosen. This means for both tests, they will be sending data of various values and sizes to the BarChart to ensure that the data is successfully passed and is displayed properly. The what-if scenario is more complicated because the user will be able to change this data multiple times so it will be more important to test. The test for the what-if scenario will be to test as many scenarios the user can use in the what-if scenario. This ensures that the data is sent to the BarChart properly and that the data is displayed properly.

Google Maps API Integration with the App

In order to test the Google Maps API with the app, the test will require using a mock location. This will enable testing the part of the app that obtains the user's location. The test will include obtaining the location from several different mock locations and selecting a location if the user denies the app getting the user's location. Testing with mock locations ensures that the app is consistent and accurate when getting the user's location.

There will be a second test for the Google MAps API by confirming that different locations are successfully passed to the GraphResults activity. This integration test for getting the chosen location will be simpler because it requires testing the retrieval of the chosen location in a single activity. This will require to two cases to test; the first will test the getting the correct chosen location and the second will test getting the correct chosen location when a new location is selected. These two tests ensures that the retrieved location is up to date on each call.

Usability Testing

Usability testing will be used to ensure that our application is easy to use and understand. We will be conducting test with a small group of individuals with an assortment of backgrounds. Our participants will be given a set of tasks to accomplish and we will watch as they set out to perform these tasks. While they are doing the tasks, we will be able to get feedback on how long it takes to perform the tasks and where or what causes them confusion. The participants will give us feedback at the end of the tests to let us know what they liked and disliked about the application. We will ask for feedback on the user interface and user experience. We will also ask for thoughts on the clarity, flow, and final thoughts on the application. The results of this testing will allow us to make changes to the application to clarify parts that may be confusing or to fix bugs not caught in the unit tests. The tasks are as follows:

1. View list of animals at your current location
2. Sort the current animals by descending mass
3. Read description of largest mammal
4. View wikipedia page of largest mammal
5. See list of historic animals at current location
6. View bar chart
7. Remove all animals greater than 15kg from ecosystem
8. Reset values
9. View current spatial map
10. View historic spatial map
11. Go back to map and choose a location in Africa
12. View current animals in specified location

This list of tasks will take the user through every aspect of the application. The layout of the tasks follows a basic use case and flow of the application that we expect to be the most common. By watching a set of users perform these tasks we can get a good idea of how the flow of the application works to people not intimately familiar with its format and will allow us to see what works well and what can be improved.

There will be a minimum of five participants. One of the participants will be a computer science student working on another capstone project. The purpose of this is that this student will be able to give us critical feedback on the app design and useful information on the technical aspects. Another participant will be a biology student who has taken ecology courses. This provide feedback on the ecological side of the application and to

get thoughts on how are results are being displayed and described. Other participants will have no speciality regarding the focus on the application. Feedback from these participants will give insight to how the application is perceived to our general audience. The important aspect of this group of participants is to make sure the layout of the application is user friendly and to ensure that the data is easy to understand and follow.

Conclusion

Performing tests and getting feedback from users is an important process in getting the application to its final stages before release. The three methods of testing allows us to take a closer look at the app to refine and fix minor problems. The unit testing is useful for going through all of the use cases and write specific tests on the methods that made these cases work. Through these tests our app will be tested with every possible input and scenario to ensure that the final product will be able to function and handle any use case. The integration testing is useful for checking that the modules worked together and pass data accurately. Testing to make sure the database works seamlessly with the app is a critical requirement in being confident in the robustness of the product.

Getting user feedback will be beneficial in getting unbiased and unique insight into how the application looks and feels to users of different backgrounds and knowledge. This will help us tweak the app and the user interface to provide the best possible user experience. The ability for our application to function seamlessly without any unexpected crashes or bugs is essential to rolling out the final product.