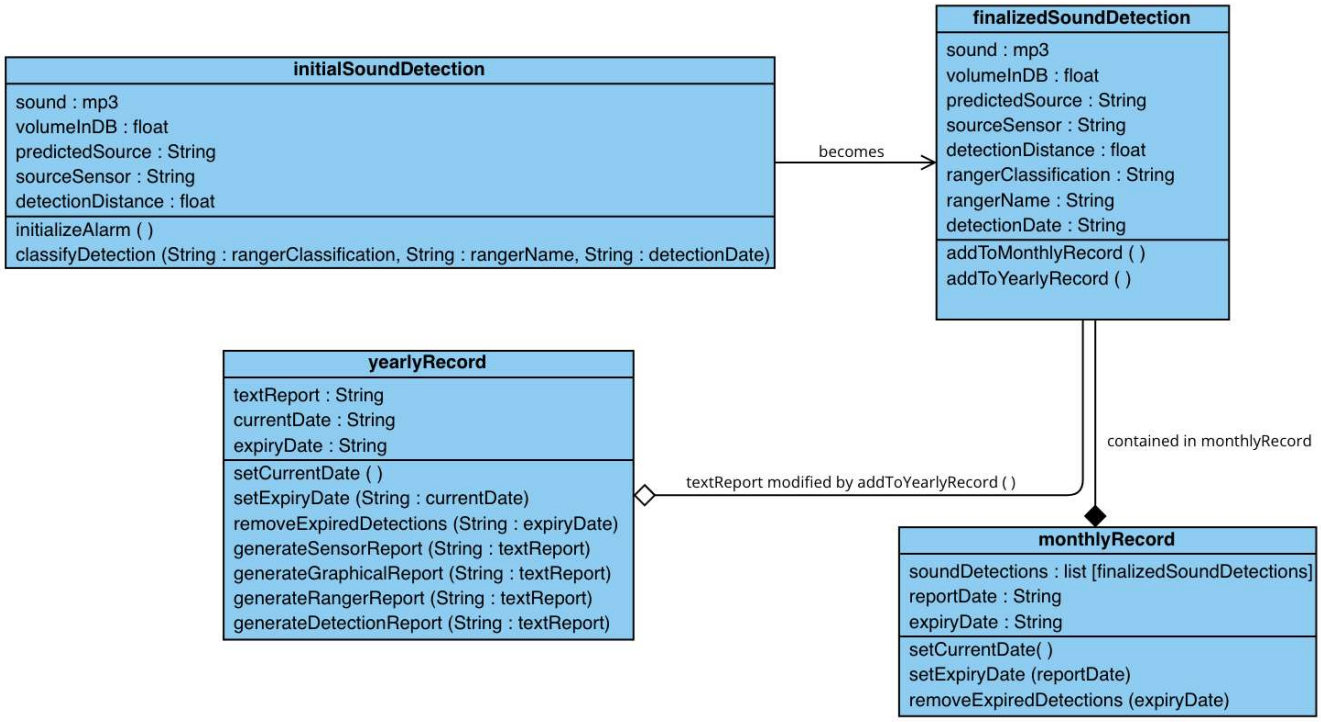
Group 9 Mountain Lion Detection System

Software Design Specification: Test Plan

Team Members: Alberto Ames, Carlos Paredes, and Bobby Tomlinson

Software Design Specification



Classes

The Group 9 Mountain Lion Detection System has the following classes: initialSoundDetection, finalizedSoundDetection, yearlyRecord, and monthlyRecord. initialSoundDetection is an object that is instantiated by the sound sensor and is transmitted to the central computer as an alert. Its main purpose is to store data regarding the detection; however, within this class, the detection initializes the alert to park rangers, and the user uses the class's classifyDetection operation to determine the threat level of the detection. Once this is done, the initialSoundDetection becomes a new object, based on the class finalizedSoundDetection. This class, once again, stores data regarding the detection. Through its operations, finalizedSoundDetection also has the ability to add itself and remove itself from the computer's month-long record and its year-long record. The class monthlyRecord stores a list that contains finalizedSoundDetection objects. monthlyRecord also routinely uses the operations setCurrentDate and setExpiryDate to update its attributes reportDate and expiryDate. monthlyRecord uses these attributes to call removeExpiredDetections and remove finalizedSoundDetections from its list. yearlyRecord, like monthlyRecord, has attributes for its expiry and current dates with corresponding routine operations to update these attributes. Additionally, it can generate a variety of reports, and contains a textReport attribute with a

summary of detections that is updated as detections add themselves to the class. Thus, initialSoundDetection is an object that eventually becomes finalizedSoundDetection through user interactions. finalizedSoundDetections access the attributes of yearlyRecord, and are contained within monthlyRecord.

Attributes

The classes of the Group 9 Mountain Lion Detection System follow the principles of Object-Oriented Programming, and as such, are primarily intended as stores of data. Thus, it is important to note the purposes of each class's attributes. First, the initialSoundDetection class makes use of mp3 audio libraries in order to store the audio from the detection within the initialSoundDetection with a type of mp3. Next, the volume of the detection, measured in decibels, is stored within initialSoundDetection as volumeInDB of float type. Next, the predictedSource of the audio, based on user-programmed audio and a corresponding user-entered string, is stored within initialSoundDetection as predictedSource of string type. Next, the sensor that detected the sound of interest is assigned a user-entered name upon installation, and when a detection occurs, the name of the sensor is stored within initialSoundDetection as sourceSensor of string type. The sensor's predicted distance (in meters) from the sound of interest is stored within initialSoundDetection as detectionDistance of type float. When initialSoundDetection is instantiated, and its alert causes a park ranger to respond, initialSoundDetection becomes finalizedSoundDetection. finalizedSoundDetection stores all of the previously mentioned attributes of initialSoundDetection, with some additional attributes due to user interactions. Supplementally, finalizedSoundDetection stores the responding ranger's classification of the threat stored as rangerClassification, which is subjectively decided by the ranger based on the audio. This variable can be one of three strings: Definite, Suspected, or False. Additionally, finalizedSoundDetection stores the user-entered full name of the responding park ranger within the variable rangerName of type string. Furthermore, finalizedSoundDetection stores the date of the detection as the variable detectionDate. This data is carefully entered by the responding ranger according to a specific format upon which the functionalities of further operations are dependent, and this data is of string type. monthlyRecord contains the necessary libraries to allow for a dynamically-created list that stores instances of finalizedSoundDetection of object type "finalizedSoundDetection." Additionally, monthlyReport also contains a reportDate attribute of type string that is updated daily by setCurrentDate, and this string represents the most recent date that the object contains detections for. Additionally, monthlyReport contains a variable setExpiryDate of type string that is updated daily by setExpiryDate, and this string represents the least-recent date that the object contains detections for. Similarly the class yearlyReport also contains attributes for currentDate and expiryDate, both of type string, both containing the most recent and least recent dates for which yearlyReport contains detections for. yearlyReport also contains a variable textReport of type string that is relatively long and contains a natural language summary of all of the detections occurring for the past year. This string is dynamically updated as new detections are added and summarized.

Operations

The Group 9 Mountain Lion Detection System contains relatively few operations, and most have to do with updating dates and removing/adding soundDetection objects to containers. initialSoundDetection has an operation that executes upon its instance of initialSoundDetection being received by the central computer, and results in flashing lights and sounds within the ranger station, controlled externally by the central computer. initialSoundDetection also contains the operation classifyDetection that takes input in the form of strings (ranger name, detection classification, detection date) from the user in order to construct a finalizedSoundDetection object from attributes of the original initialSoundDetection, along with new, user-entered attributes rangerClassification, rangerName, and detectionDate. finalizedSoundDetection has the operations addToMonthlyRecord and addToYearlyRecord. Both of these operations are executed once the finalizedSoundDetection is instantiated. addToMonthlyRecord accesses the monthlyRecord object's list attribute "soundDetections" in order to add finalizedSoundDetection to the list within this monthlyRecord. addToYearlyRecord converts the attributes of finalizedSoundDetections into a text summary (1-2 sentences) of type string that is prepended to the larger string attribute textReport of object yearlyRecord. monthlyRecord has the operation setCurrentDate that accesses the central computer information to determine and store the current date. monthlyRecord also has the operation setExpiryDate with the input parameter of currentDate that creates a string expiryDate that represents the date 30 days prior to the currentDate. monthlyRecord also has the operation removeExpiredDetections with the input parameter expiryDate that iterates through the list soundDetections and checks each the detectionDate of each finalizedSoundDetection, and if there is a match to the expiryDate, the corresponding finalizedSoundDetection is removed from the detectionDates list. yearlyReport also has the operation setCurrentDate that functions identically to the setCurrentDate operation of monthlyReport. yearlyReport has the operation setExpiryDate that sets the string expiryDate equal to the date that is 365 days prior to currentDate. yearlyReport has the operation removeExpiredDetections with the input parameter expiryDate in order to remove portions (sentences) of the string textReport that correspond to a detection (or detections) at the expiryDate. yearlyReport has the operations generateSensorReport, generateGraphicalReport, generateDetectionReport, and generateRangerReport. These operations each take in the textReport attribute as an input parameter to generate the reports detailed in bullets 1-4 of the System Description - generateDetectionReport corresponds to bullet 1, generateSensorReport corresponds to bullet 2, generateGraphicalReport corresponds to bullet 3, and generateRangerReport corresponds to bullet 4.

Verification Test Plan

Unit Testing

The first test set will test the function classifyDetection. The test will involve playing audio near one of the sensors and seeing if the function successfully identifies the sound as a threat or non-threat. To achieve this, we should play a variety of different sounds at different volumes and different distances from the sensor. One such example will involve playing audio of a mountain lion at a volume of 3 dB, 1.5 meters away from the sensor. In addition, we should include test cases with no sound at all to ensure that the function correctly identifies silence as a non-threat. This can be achieved by simply placing the sensor in a quiet room and ensuring that the function does not produce any false positives.

We should also test the detectionDate variable passed in. We can start with a valid date, like 3/23/2023, but we should also test a nonsensical date such as 0/00/0000. If the second date is passed in, we should get an error along the lines of: “Error: date invalid.”

Functional Testing

Our first set of functional testing will verify the proper integration of the initialSoundDetection class and the finalizedSoundDetection class via the operation classifyDetection. This test first assumes that finalizedSoundDetection and yearlyRecord are properly integrated. In order to truly verify this assumption, the integration will need to be verified with a future test set not currently included in this test plan, but will most likely be included as a clear-box test as future iterations of this software system are developed. To test the integration of initialSoundDetection and finalizedSoundDetection, we first physically play audio of a mountain lion at a volume of 2 dB approximately 3 meters away from a sensor known by the system as sensor C. Once this detection is initialized and the alarm is initialized, the user will be prompted by the UI of the central computer to review the data, including the sound, volume, predicted source animal, the sensor that initialized the detection, and the detection’s distance from the sensor. The data should all correspond to the 2dB mountain lion sound that was detected 3 meters away from sensor C. Next, the ranger is prompted to provide a classification, their own full name, and the date of the detection, which will all be entered and stored as strings. For the first test in this set, the ranger will enter a classification of “False” and the name “John Smith.” For this test, the operation classifyDetection should throw an error message to the user saying “Missing detection date! Please try again.”. If the operation does not display this error message, then the finalizedSoundDetection most likely was constructed with a null detectionDate attribute, and the error handler of the classifyDetection operation will need to be reviewed. However, if the error message was displayed, then the test was successful, and the next test in this set can begin. For the next test, the user will this time enter a classification of “False,” a

name of “John Smith”, and a date of “3/21/2023.” For a successful test, no error message gets thrown, and the finalizedSoundDetection object gets instantiated, and becomes a part of the storage of the central computer. This is tested by calling generateSensorReport for sensor C to view the detections at sensor C, and if the report’s most recent detection includes a detection of volume 2dB at sensor C from a mountain lion 3 meters away, classified by John Smith as false on 3/21/2023, then the classifyDetection operation works effectively. A successful test means that the detections are being properly classified and modified after user interactions from the ranger. A failing test means that the detections are not properly modifying and saving the detections according to user input.

Next, our second set of functional testing will verify the proper integration of the yearlyRecord class and the finalizedSoundDetection class. This test set is to be begun after the previously mentioned first set has been completely tested and verified. First we will begin by playing a 7 dB mountain lion sound 0.5 meters away from sensor B. The detection wll be classified as “Definite” by the ranger “Jane Doe” on the date 3/22/2023. Next, the ranger computer’s date will be manually adjusted to 4/24/2024. If the operation removeExpiredDetections in yearlyRecord worked properly, then when generateSensorReport is called for sensor B, the report should be empty, and should not include the detection that was just classified by Jane Doe as “Definite” on 3/22/2023 with a 7dB mountain lion sound 0.5 meters away from sensor B. A successful result for this test would have the output of an empty Sensor Report for Sensor B. A failing result for this test would have a report with detections in it, and/or would include Jane Doe’s 3/22/2023 “Definite” detection that was set up for this test. A failing test means that yearlyRecord’s method of adjusting expiry dates and removing expired detections is flawed and stores detections for an excessive amount of time, causing wasted storage. A successful test means that the yearlyRecord class correctly and efficiently stores the correct detections.

System Testing

Test 1: Detection and Alert Messaging

1.) Place noise detection sensors within an area of 5 square miles.

2.) Program the device to detect mountain lion noises.

3.) Create a mountain lion like noise within said area, make sure the noise is strong enough to break the threshold for alerting.

4.) Check if an alert message is sent to the controlling computer, including type of noise detected, strength of detected noise, and the location of detected noise within 3 meters.

5.) Check that the alarm sounds on the controlling computer, and that it continues until the ranger turns it off.

6.) Check that if another noise is detected at a different location, the alarm will sound again.

Test 2: Reporting and Classification

1.) Set off multiple simulated mountain lion noises within the detection area.

2.) Check that the controlling computer saves all mountain lion alerts received with the last 30 days and a summary of alert information for data older than 30 days but received within one year.

3.) Check that the ranger can classify each alert as definite, suspected, or false, showing the probability that a real mountain lion was detected.

4.) Check that the control program allows the ranger to request several reports:

a.) **Report #1**: Shows all mountain lion detections by date detected and by classification (definite, suspected, or false).

b.) **Report #2**: Shows detections at a specific sensor location.

c.) **Report #3**: Shows detections on a map of the park and areas within 2 miles of the park.

d.) **Report #4**: Shows a detection classification by ranger.

5.) Check that the system can be easily reconfigured for other parks in the state of California.

Overview

Test 1: Detection and Alert Messaging

The goal of this test is to ensure that the noise detection sensors are functioning properly and are able to detect mountain lion noises within a specified area. The test involves emitting a simulated mountain lion noise and verifying that an alert message is immediately sent to the controlling computer, and that the alarm on the computer activates and continues to sound until manually turned off by a ranger. The test also checks that the system can detect multiple noises at different locations and sound an alarm each time a noise exceeding the threshold is detected.

Test 2: Reporting and Classification

The goal of this test is to ensure that the controlling computer is able to properly store and organize all mountain lion alerts received, and that rangers are able to classify each alert as definite, suspected, or false based on the probability that a real mountain lion was detected. The test involves simulating multiple mountain lion noises at various locations and verifying that the controlling computer can store and organize all alerts received within the last 30 days. The test also checks that rangers can request various reports on the alerts received, including a chronological list of all detections by date and classification, a list of all detections at a specific sensor location, a map displaying all detections within the park and areas within 2 miles of the park, and a classification breakdown of detections by ranger. Finally, the test confirms that the system is easily adaptable for use in other parks throughout California.