Mountain Lion Detection System Software Design 2.0

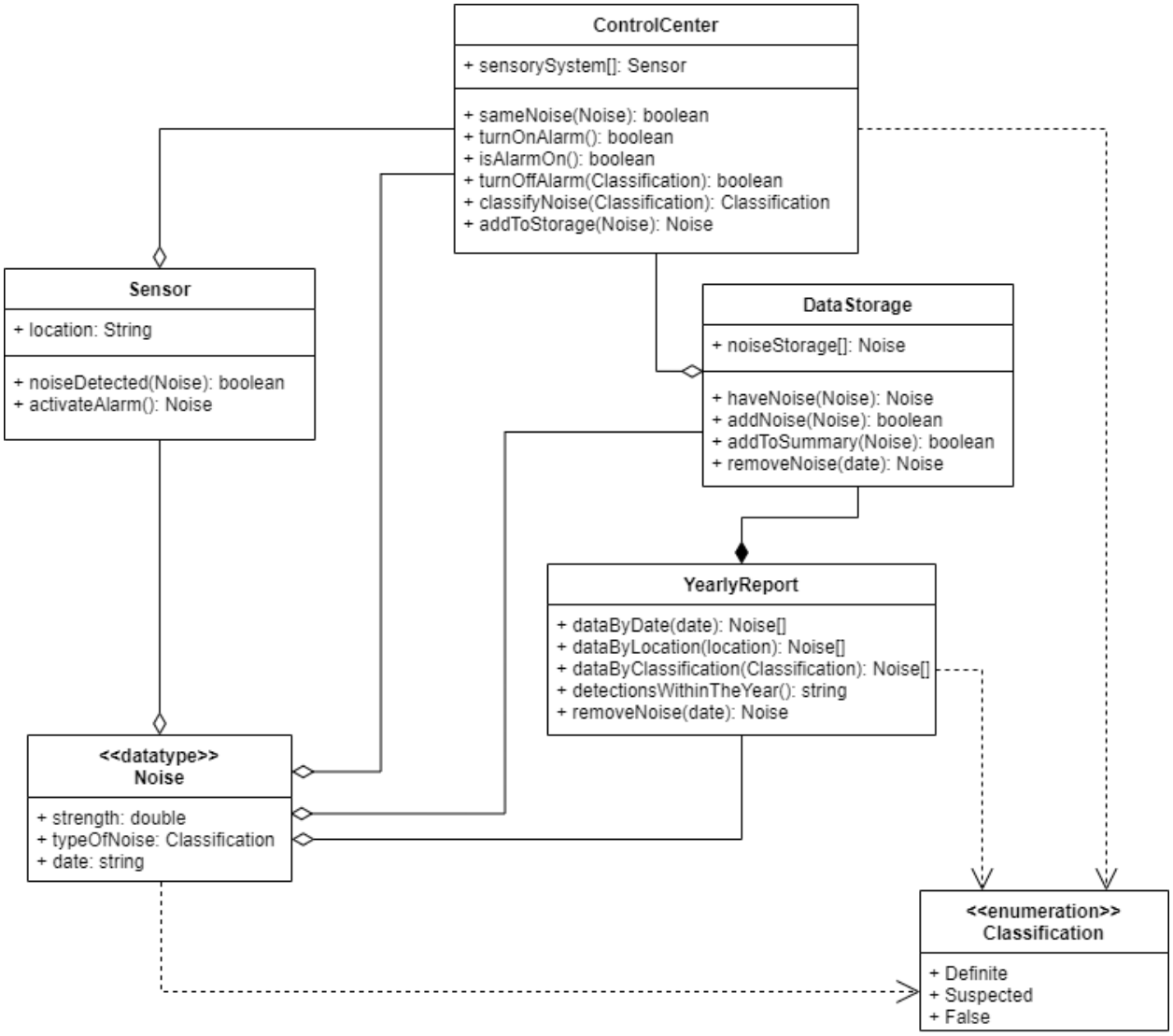
Prepared by: Jonathan Trinh, Christopher Diep, Jasmine Nguyen, Simon Rofaeel

November 30, 2021

System Description

The Mountain Lion Detection System belongs to the San Diego County Parks and Recreation Department and is based on the development of the animal detection system by the Animals-R-Here company. The purpose of the system is to detect all mountain lions and other types of animals found around the parks. Within the system, there are noise detection sensors in a five square mile radius, as well as a detection classifier for the various types of animal noises. There are controlling computers located in the park ranger stations and all park rangers have access to them. Within the system, alert messages can be requested to be sent to the controlling computers with the strength, location, and type of noise that is detected. The control program on the controlling computers will also sound an alarm whenever there are alerts received within the animal detection system. The system will be able to record the location, date, and time of all mountain lion noise detections, and allow rangers to request yearly reports of this information, along with the specific sensor location and the classifications made by the rangers from the park.

Architectural Diagram



USE CASE DIAGRAM

Mountain Lion or
Other Potential Noises

System sensor detects noise

Same noise?

Yes

Does not activate alarm

No

System records strength, location, time, and date

System activates alarm

Data storage

Ranger

Turn off alarm

Classify noise

System records classifcation

Request reports

Yearly summary

30 days have passed

System deletes 30 day old records

Class Description

ControlCenter: The controlCenter class provides functions for the user to access parameters and states, and handles a number of conditions that may occur. The class sends all the information the noise detectors will process.The controlCenter class inherits from all of the other classes. This information updates when the backend data is changed. Furthermore it holds information if it is the same noise detected or a different one in case there will be another situation in which another mountain lion is detected in a separate location and the classifications of the noises.
This class allows the user or ranger to be able to:

- Turn the alarm on and off
- Classify each alert as definite, suspected, or false
- Save all mountain lion alerts within 30 days
- Save a summary of alerts for data older than 30 days but received within 1 year
- View/edit/delete alerts
- Request reports of all mountain detections by date, location, and classifications by rangers.
    - ➔ ***Attributes:*** *sameNoise(Noise), classifyNoise(Classification).*
    - ➔ ***Operations:*** *sensorySystem[], turnOnAlarm(), turnOffAlarm(), addToStorage(Noise).*

Sensor: Alarms are only activated when the sensor detects a noise classified as a mountain lion within the range of the detector. The detectors have a range of five square miles. The sensor class has an association to the controlCenter class and noise class, therefore it will send an implementation of activateAlarm() when a noise is detected. Once the sensor detects a noise it shall be able record the location within 3 meters of the sensor as a string to send to the control center for the database. The alarm will sound until a ranger turns it off from the control center, and will not activate again until a different noise is detected in a different location.

- ➔ ***Attributes:*** *location, noiseDetected()*
- ➔ ***Operations:*** *activateAlarm()*

Noise: The noise class is a data type class that holds a collection of attributes shared between all classes and abstracts the operation of the sensor. The class shall be able to record the strength of the noise as a double, the classification of the noise, and record the date of when the noise was detected. It is an aggregation to dataStorage class, yearlyReport class, and controlCenter class.

- ➔ ***Attributes:*** *strength, typeOfNoise, date*

DataStorage: This class holds data about different types of noises classified by the user/ranger in the controlCenter class to reduce error and aggregates from the controlCenter class. It holds the date, time, and location information of each mountain lion noise detection sent from the

sensor and noise class. This class allows the user to edit/add/remove noise alert information for the database and adds it to the summary for yearly reports.

- ➔ ***Attributes:*** *haveNoise(Noise)*
- ➔ ***Operations:*** *noiseStorage[], addNoise(Noise), addToSummary(Noise), removeNoise(data)*

YearlyReport: The yearlyReport class holds properties from the noise class, sensor class, and classification class, and is a composition of the dataStorage class as it cannot exist meaningfully without it. This class allows the user to request several reports: a report of all mountain lion detections by date and classification(definite, suspected, or false), a report of all detections by sensor location, a report of each detection classification by ranger, and a yearly summary of all mountain lion detections.

- ➔ ***Attributes:*** *dataByDate(date), dataByLocation(location), dataByClassification(Classification), detectionsWithinTheYear()*
- ➔ ***Operations:*** *removeNoise(date)*

Classification: An enum class that contains three constants, “Definite”, “Suspected”, and “False”. The returned enum object works as an iterator for the activateAlarm() method as it can sequentially fetch elements stored in the typeOfNoise object. Only rangers are able to classify an alert as definite, suspected, or false to indicate the likelihood of any mountain lions that are within 5 square miles of the area of the sensor. The controlCenter class, yearlyReport class, and Noise class are dependent on the classification class.

Attributes and Operations

Control Center

Attribute / Operation	Type	Description
sensorySystem[]	Sensor	The system employs a sensor to indicate whether or not an animal is detected.
sameNoise(Noise)	Boolean	The system compares newly detected noises against animal detection classifications to determine if new noise has been detected before; returns true if the same noise is detected and false otherwise.
turnOnAlarm()	Void	The system shall turn on the alarm if

		noise is detected and return the control to the caller.
turnOffAlarm()	Void	The system shall turn off the alarm when prompted by the user.
classifyNoise(Classification)	Classification	The user shall classify the detected noise-type.
addToStorage(Noise)	Noise	The system shall add newly detected noise to the system storage.

Sensor

Attribute / Operation	**Type**	**Description**
location	String	The system shall display the location of the detected noises.
noiseDetected()	Boolean	If a noise is detected, sensors will alert the system; returns true if noise is detected and false otherwise.
activateAlarm()	Noise	If a noise is detected, the system will activate the alarm; returns true if the alarm is activated and false otherwise.

<<datatype>> Noise

Attribute / Operation	**Type**	**Description**
strength	double	The system shall report the detected noise strength level in terms of double.
typeOfNoise	Classification	The system shall classify the type of noise detected.
date	String	The system shall display the date of the detected noise.

DataStorage

Attribute / Operation	Type	Description
noiseStorage[]	Noise	The system shall have a declared noise storage.
haveNoise(Noise)	Noise	The system determines if a noise is detected.
addNoise(Noise)	Boolean	The system shall add detected noises to the system's storage; return true if noise data is added to the system and false otherwise.
addToSummary(Noise)	Boolean	The system shall add detected noises to the system's storage summary; return true if noise data is added to the system summary and false otherwise.
removeNoise(date)	Noise	The system shall remove detected noises older than 30 days from the system's storage summary.

YearlyReport

Attribute / Operation	Type	Description
dataByDate(date)	String	The system shall allow for yearly reports, filtered by date of noise detection and data logging.
dataByLocation(location)	String	The system shall allow for yearly reports, filtered by location of noise detection and data logging.
dataByClassification(Classification)	String	The system shall allow for yearly reports, filtered by classification of noise detection and data logging.
detectionsWithinTheYear()	String	The system shall report the noise detections received within the year.
removeNoise(date)	Noise	The system shall remove noise detection data from the yearly report

		after the data has surpassed a year.

<<enumeration>> Classification

Attribute / Operation	Type	Description
Definite	N/A	Within the constructor of various attributes, if a noise is definitely detected, the system will classify as “Definite”
Suspected	N/A	Within the constructor of various attributes, if a noise is suspected, the system will classify as “Suspected”
False	N/A	Within the constructor of various attributes, if a noise is not detected, the system will classify as “False”

Test Case Purpose

The purpose of this Mountain Lion Detection System Software Test Plan is to describe the scope, approach, resources, and schedule of testing activities for this project. It gives an integrated view of the test activities and defines the overall testing requirements. This document will describe how:

- What will be tested
- How testing will be performed
- Personnel conducting the testing based on the identified risk level

Test Scope

The scope of testing is to perform tests at different stages of development. The following tests considered are:

- Unit testing
- Integration testing
- System testing

Item Pass/Fail Criteria

The testers will mark the status of a completed test within the system as “Pass” or “Fail” with defining when an item or testing effort demonstrates correct adherence or unacceptable results.

Test Cases

Test Case No.	Test Case Description/Actions	Inputs	Expected Outputs	Test Result	Pass/ Fail

1	**Unit Test**
	
	Check functionality of pulling the date from the system; system shall reflect an accurate date of detected noise.
		bool testingDataByDate() {
		std::string date1 = "09/14/21";
		std::string date2 = "06/15/20";
		std::string date3 = "no";
		
		if(YearlyReport.dataByDate(date1) == NULL) {
		return false;
		}
		
		if(YearlyReport.dataByDate(date2) != NULL) {
		return false;
		}
		
		if(YearlyReport.dataByDate(date2) != NULL) {
		return false;
		}
		return true;
		}
			This should all return true.	Print “TRUE” to specify the noise has been date-logged into the system.	Pass

2	**Unit Test** Checking functionality of pulling the location of the pin-pointed noise from the system; system shall reflect an	bool testingDataByLocation() { std::string location1 = "5A"; std::string location2 = "2D"; std::string location3 = "N/A";	This should all return true.	Print “TRUE” to specify the newly detected noise location has been logged into the system.	Pass

	accurate location of detected noise.	if(YearlyReport.dataByLocation(location1) == NULL) {
		return false;
		}
		
		if(YearlyReport.dataByLocation(location2) != NULL) {
		return false;
		}
		
		if(YearlyReport.dataByLocation(location3) != NULL) { //Location does not exist
		return false;
		}
		}			

3	**Interface Test**
	
	Checking to see if a noise is detected, if so the control center receives a notification and the alarm system turns on.
		bool testingNoiseDetection() {
		Noise test1 = new Noise(2.5, "03/05/13");
		Sensor.noiseDetected(test1);
		if(!ControlCenter.isAlarmOn()) {
		return false;
		}
		return true;
		}
			This should all return true.	Print “TRUE” to specify the control center receives noise detection and turns the alarm on.	Pass

4	**Interface Test**
	
	Checking to see if newly detected noise is logged into the system database.
		bool testingAddNoise() {
		Noise test1 = new Noise(2.5, "05/12/14");
		Noise test2 = new Noise(1.3, "05/15/14");
		
		DataStorage.addNoise(test1);
		
		If(!DataStorage.haveNoise(test1)) {
		return false;
		}
			This should all return true.	Print “TRUE” to specify the newly detected noise location has been logged into the system.	Pass

		If(DataStorage.haveNoise(test2)) {
		Noise that shouldn't exist.
		return false;
		}
		return true;
		}			

5	**System Test**
	
	The system is put under test to ensure detected noise performs all functions required. The system turns off the alarms when rangers classify the noise and notify the system to turn the alarm off, the newly detected noises are added to the database and yearly report. The system is being tested to determine if all functions are working properly.
		bool systemTesting() {
		Noise[] testingNoises = new Noise[4]; //four test noises
		testingNoises[0] = new Noise(3.5, "04/15/21");
		testingNoises[1] = new Noise(1.4, "06/17/21");
		testingNoises[2] = new Noise(4.2, "11/05/21");
		testingNoises[4] = new Noise(); //Default noise which sets every parameter to zero or null
		
		Sensor.noiseDetected(testingNoise[1]); //Sensor detects a noise and activates the alarm for the system.
		if(!ControlCenter.isAlarmOn()) { //Checks to see if alarm turns on
		return false;
		}
		
		ControlCenter.turnOffAlarm(Suspected); //When you turn off alarm you should be prompted to classify the noise
		
		Noise[] classifiedNoises = dataByClassification(Suspected); //Let's assume the user inputted SUSPECTED
			This should all return true.	Print “TRUE” to specify all functions are working properly.	Pass

		bool foundNoise = false;
		for(int i = 0; i < classifiedNoises.length< i++) {
		//Check the list of classified noises to see if the noise we tested is there
		if(classifiedNoises[i] == test1) {
		foundNoise = true;
		break;
		}
		}			

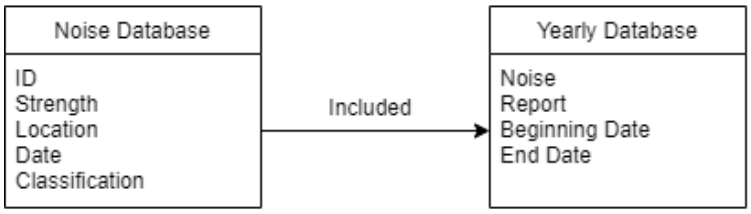
Data Management Strategy Purpose

The purpose of this Mountain Lion Detection System Data Management Strategy is to describe how the team will manage the data during the project. The strategy describes the data that will be acquired, how it will be managed, described, stored, handled, and what standards we will use during and after the project.

Scope

The team will be using a SQL style database to represent the data of the software system because it is relational and table-based. SQL will be beneficial to the system because they have a well-designed pre-defined schema for structured data for software systems like this and have vertical scalability. There will be two databases integrated into the system: Noise Database and Yearly Database. The data dictionary is an inventory of data elements in the database model with a detailed description of its format, type, description, relationships, and usage.

Data Management Strategy



Data Dictionary
Noise Database: Database of noises Type: Table Description: Noise information Primary Index: ID Secondary Index: Location
Name: ID Type: attribute/integer Description: unique identifier for each noise. Expected values 0-9 Optional: no
Name: Strength Type: attribute/integer Description: strength of noises detected Expected value: 0-5 Optional: No
Name: Location Type: attribute/alphanumeric Description: Location noises are detected Expected length: 1-20 Optional: No
Name: Date Type: attribute/alphanumeric Description: Date when noises are detected Expected length: 8 Optional: No
Name: Classification Type: attribute/enum Description: Classification of Noises Expected enum: "Definite" "Suspected" "False" Optional: No
Name: Yearly Database Type: Table Description: Database report to users Primary Index: Noise Secondary Index: Beginning Date
Name: Noise Type: Noise Description: Noises Expected Amount of Noises: 1-999 Optional: No
Name: Beginning Date Type: attribute/alphanumeric Description: Earliest Date of Reports Expected length: 8 Optional: No
Name: End Date Type: attribute/alphanumeric Description: Latest Date of Reports Expected length: 8 Optional: No