Software Design Specification

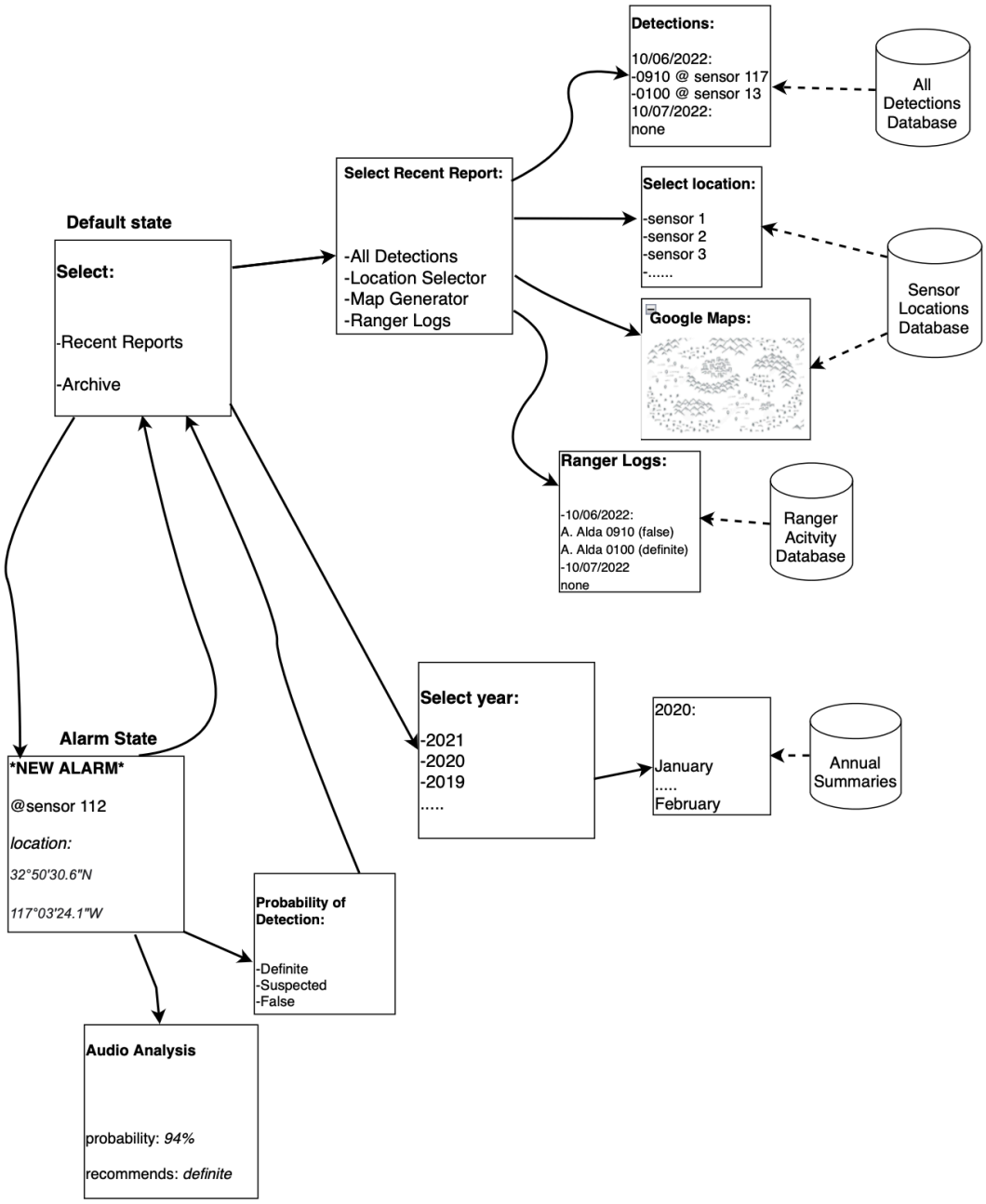
Mountain Lion Detection System

Ali K.
Chris F.
Zhiwei H.

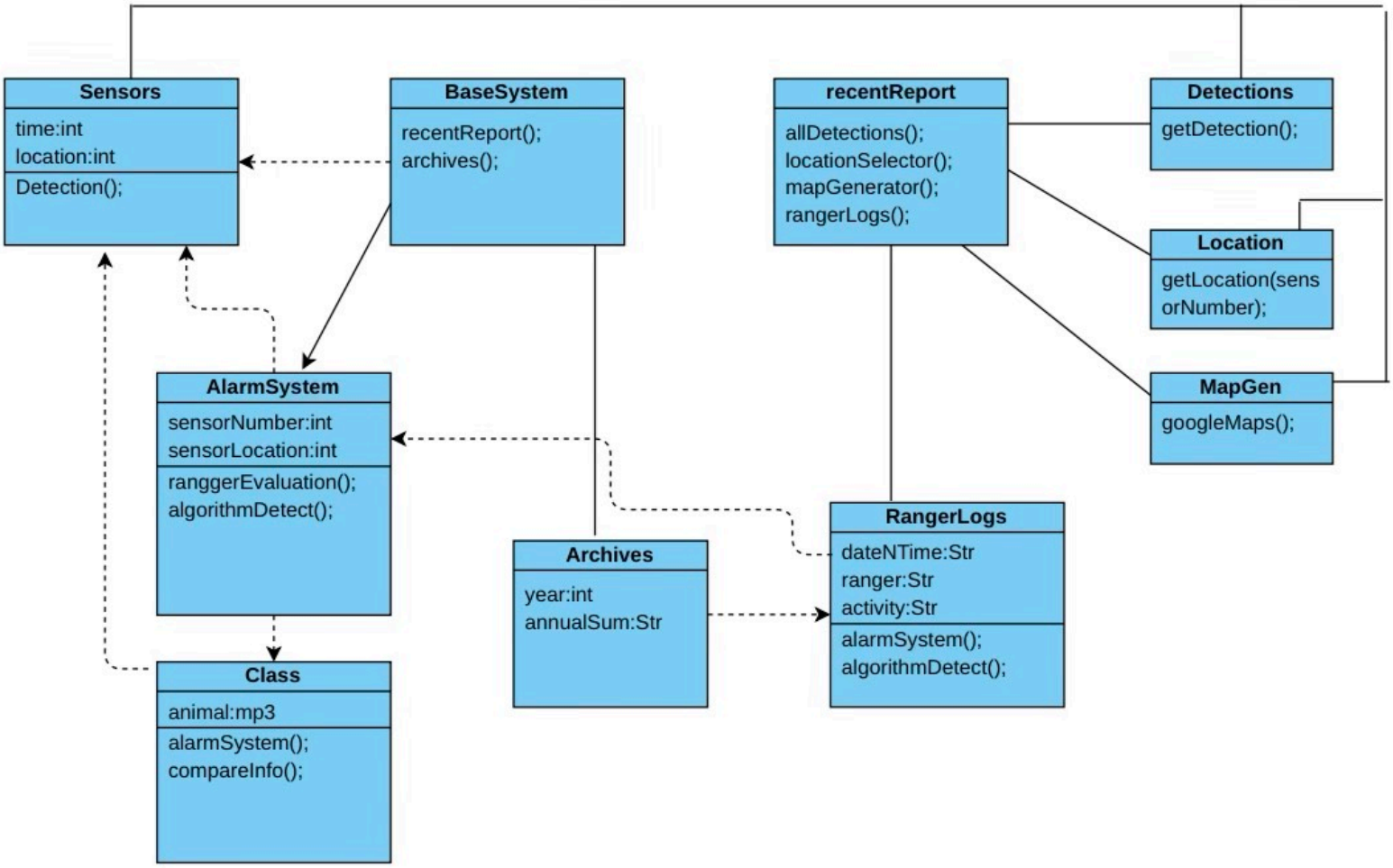
System Description

This system is designed to provide rangers with quick and reliable alerts about mountain lion activity within their parks. At the ranger station, users can receive alerts sent from one of hundreds sensors distributed throughout the park. The rangers' sole responsibility is to assign a probability classification to the detections as they are received by the system. The system also runs an algorithm automatically that generates a recommendation for the ranger based on the newly received audio file attached to the alert. Once the ranger has classified their new detection, the system reverts back to its default state which allows the user to navigate through all the past detections. Users can also filter the data by location or view the detections on a map of the park. Older reports from previous years can be accessed through the archives to generate annual summaries. For accountability, Ranger logs are kept to track all classifications performed by each ranger by date.

Architectural Diagram



UML Class Diagram



The following UML diagram is created to visually illustrate the implementation of the Mountain Lion Detection System. Starting off with the class BaseSystem, this serves to contain and direct to the main components and functions. The BaseSystem class contains two operations branching off to two different classes. The first operation is recentReports which directs to the RecentReports class. The second operation is archives which directs to the Archives class. Beginning with the RecentReports class, this class serves to store all the reports, and contains four optional operations. The first option is the allDetections operation which directs to the Detections class. Inside the Detections class there is a getter method called getDetection() which retrieves its

information from the operation called Detection present inside the Sensors class. The second operation is the LocationSelector which directs to the Location class. The Location class contains the getter method called getLocation with a parameter for the park Ranger to manually input a specific sensor number. The third operation is MapGenerator which directs to the MapGen class. The MapGen class contains the method googleMaps which is an API call allowing the user to check detection by the map. The fourth operation is rangerLogs which directs to the RangerLogs class. The RangerLogs class contains three attributes that are all string variables. These three attributes are called dateNTime, ranger, and activity. There is also an operation called alarmSystem inside the RangerLogs class which directs to the AlarmSystem class. Moving onto the Archives class, this class serves to provide yearly data and contains two attributes. The first attribute, year, is represented by an integer variable. The second attribute, annualSum, is represented by a string variable. Moving onto the Archives class which is the second branch of BaseSystem, the class Archives contains two variables. Year, which is an integer variable, and annualSum, which is a String variable. The variable annualSum provides a general overview specified by month of the detections that occurred throughout the year.

The AlarmSystem class contains two variables called sensorNum and sensorLocation which are both of type integer. This class also contains two operations. The first operation is rangerEvaluation. The dotted arrow from the class RangerLogs that is pointed at the AlarmSystem class indicates the dependency that RangerLogs has on AlarmSystem. The second operation is called algData(). The algData operation retrieves its info from the AlgorithmData class. The AlgorithmData class contains one attribute called animal, and two operations called alarmSystem and CompareInfo. The AlgorithmData class retrieves its information from the Sensors class and then using its animal mp3 attribute and CompareInfo operation it compares audio files to determine the severity of the threat. Which is why there is a dotted arrow from the AlgorithmData class going to the sensors class to indicate this dependency. If determined to be a serious threat then the method alarmSystem is triggered, which is why there is a dotted arrow from the AlarmSystem class going to the AlgorithmData class to indicate this dependency.

Development Plan & Timeline

This project will be developed by a team of seven highly experienced python programmers and data scientists. The estimated time required to deliver an alpha version of this system would be 4 months and at least six months for the final product.

A small group of three will be assigned the sole task of developing the algorithm needed to provide rangers with recommendations regarding the classification of new threat detections. A database of mp3 files containing animal noises will be purchased from a third-party company based in Chicago. This will be used to allow the algorithm to compare newly-received audio files transmitted from the sensors to those stock

animal noises. Afterwards, an evaluation is presented to the user (ranger) regarding the probability of the threat. This is a highly elaborate and specialized component of the system which should take at least four months to develop.

The remainder of the team will be working either individually or in close pairs to develop some of the simpler, less complex functions. One individual will be responsible for collecting sensor data and compiling them into the three databases: *All detections, sensor locations, and Annual summaries.* One pair will work closely together in designing an easily navigable user interface for the rangers. Their responsibilities extend to developing the main functions as well since they are fairly straightforward and rely on the work of their teammates to do most of the heavy-lifting. This should primarily involve calling on algorithms and retrieving from databases. The final member will be responsible for constant testing and code review.