“Unstable Bluff” Detection System Design Specification and Test Plan

Prepared by: “The Bluffers” - Samantha James, Joshua Olaoye, Antonio Rivera

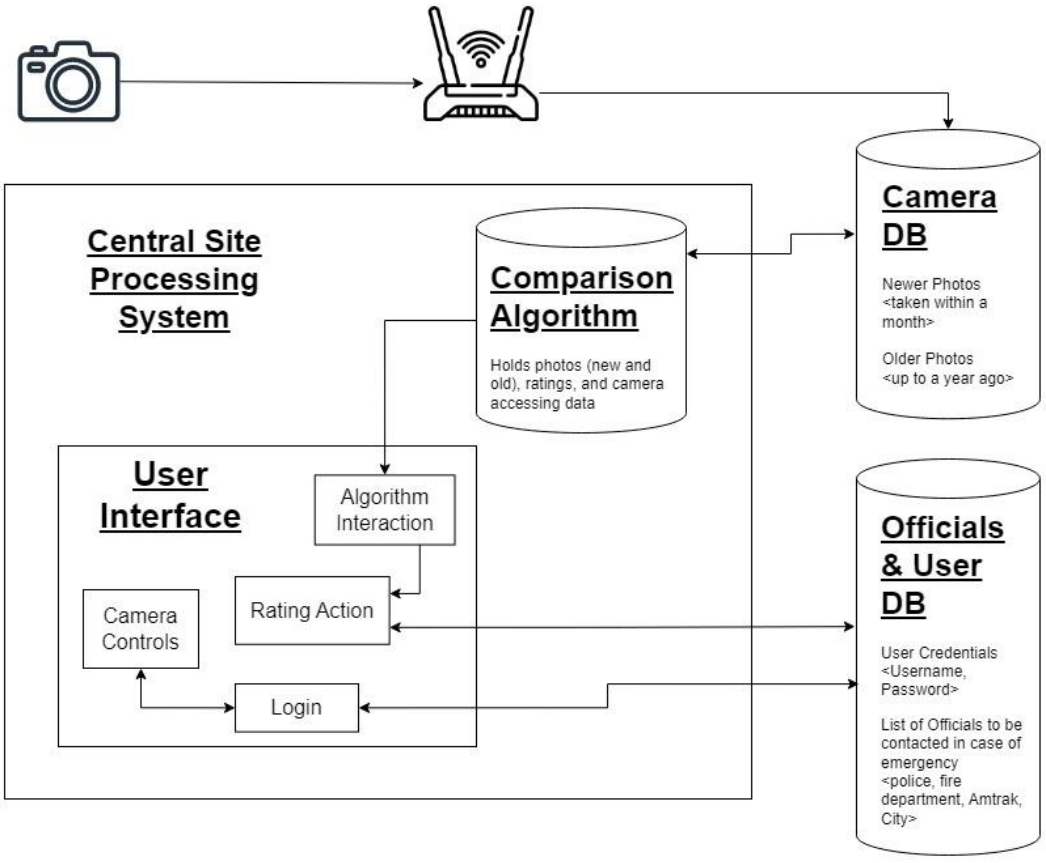
March 24, 2023

System Description

There are unstable bluffs, which are steep cliffs or banks, in the city of Del Mar along California's southern coast. A heavily trafficked train's tracks are located on a portion of these bluffs, which is very dangerous if these bluffs' stability is not monitored, potentially leading to injury and/or death of those on the train as well as people on the beach below the bluffs. This camera-based monitoring system's purpose is to constantly monitor the bluffs and detect when areas become unstable and alert authorities so that safety precautions can be implemented. This document will outline the software and architecture necessary for designing and later implementing this bluff monitoring system, which is divided into sections of software architecture overview and development plan and timeline.

Software Architecture Overview

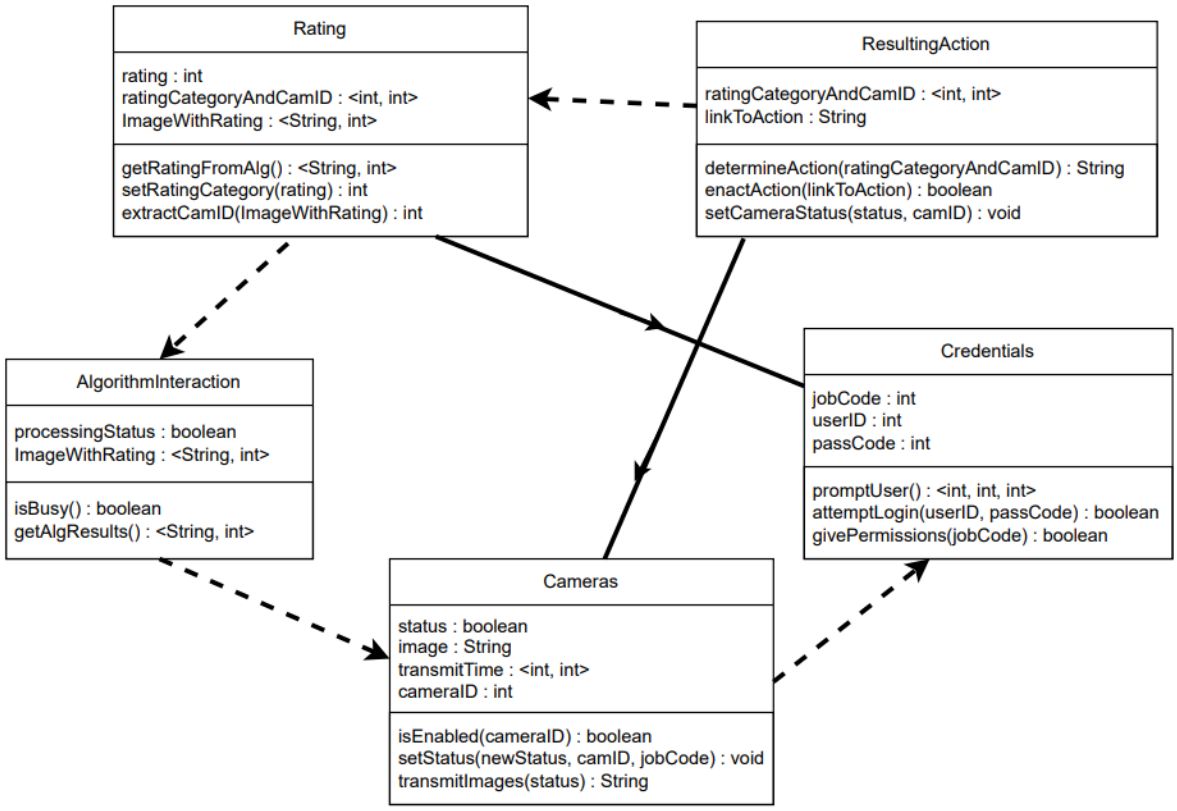
Software Architecture Diagram



Software Architecture Design Description

In this SWA diagram, a clear picture of how the system will be implemented is on display. Starting with the camera icon, this represents the six cameras placed along the cliffside, monitoring the change in stability. These cameras are then uploaded via Wifi to the nearby router, which transfers the data into the Camera Database. This database is what holds all the photos, new and old, and will be accessed by the Central Site Processing System. This system has a Comparison Algorithm implemented to extract the data from the Camera Database. This algorithm holds the photos, ratings, and camera accessing data, along with actually assigning the rating. The output of this algorithm is then sent to the Algorithm Interaction feature of the User Interface. It is from here that the user can see the photos with the rating attached to them. Depending on the rating, it is then sent to the Rating Action section of the system, where it will access the Officials and User Database, which contains the contact information of important officials in the area that need to be contacted if there is going to be a slide. There are also camera controls which interact with the login feature, which access the the Officials and User Database to see if the user who is attempting to log in has valid credentials to access the system. It is sort of daisy chained in the sense that in order to access the camera control feature, it first needs to interact with the login feature, which needs to interact with the database. The reason why the Rating Interaction does not require login access is because if the rating is high enough, it should show a warning regardless if anyone is signed in. This is prevent disaster from occurring because a user is having trouble logging in.

UML Class Diagram



Class UML descriptions

This UML class diagram depicts the classes and associations between them that we need to design and begin to implement the bluff detection system. The Credentials class involves verifying qualified officials and maintenance workers' user logins and determining what actions they are allowed to do in the system. The promptUser() function collects the potential user's jobCode, userID, and passCode, which are all integer values, sets the fields equal to the inputs, and returns them as a triple, meaning all of those integers are connected to each other. The attemptLogin() and givePermissions() functions use the inputted credentials to determine whether the user should have access to the system and if yes, then what actions is that user allowed to perform within the system. The Cameras class has variables to hold the boolean status of the camera, whether it is enabled or disabled, the image in a String as a URL that includes data like cameraID and transmitTime attached to it within the String. The isEnabled() function uses the cameraID and returns a boolean as to whether that camera is enabled or disabled, of which this status is updated through the setStatus() function which requires a jobCode to ensure the proper users are doing this action. The transmitImages() function works if the status is enabled for a camera on which the function is called and if enabled, returns a String with the image URL. The AlgorithmInteraction deals with the interactions between the system and the algorithm that runs the images through comparison tests and determines their rating. There is a variable for processing status, an integer with precoded values for on, off, in progress of processing, and receiving/transmitting images so that the system only accesses the algorithm when it is properly available and this integer is returned from the isBusy() function. The imageWithRating pair of a String and an integer hold with data returned from the getAlgResults() function that accesses the String of the new image URL from the camera along with the rating that the algorithm assigned it. The Rating class separates the important information from the image URL and keeps it attached to the rating. The getRatingFromAlg() function accesses the data from the AlgorithmInteraction class and separates the rating from the pair in order to be accessed separately for other purposes. The setRatingCategory() uses the integer rating and determines what category of severity it falls into, which will later be used for what consequential action must be taken. The extractCamID() function takes in the ImageWithRating and copies over the cameraID integer value from within the String image URL to know which section of bluff it is referring to when it gets combined with the rating category into a pair. The ResultingAction class uses variables for the ratingCategoryAndCamID pair of integers and a linkToAction String. The determineAction() function uses the rating category to return the String action that must be implemented based on the severity of the situation determined by the comparison algorithm. This returned String populates linkToAction and is pulled from a list of predetermined links for different rating categories. The disableCamera() function sets the camera with the CamID from the data pair to disabled, so it sets their status to false. The enactAction() function opens the linkToAction which typically includes local officials' contacts in order to send emergency alert messages, then the function returns true once the action has been completed. These classes are interconnected and many of them rely on each other for functionality. The Rating class has an association to the Credentials class since a user must have proper credentials in order to set and access ratings, displayed by a solid line in the diagram. Also, the ResultingAction class has an association with the Cameras class since there

is a function to disable cameras. Finally, there are many dependencies between the classes, which are displayed by dotted lines with arrows in the diagram. ResultingAction uses data from Rating, Rating uses data from AlgorithmInteraction, AlgorithmInteraction uses data from Cameras, and Cameras uses data from Credentials.

Development Plan and Timeline

In order to develop our “Unstable Bluff” detection system, we will have to create our central site processing system, including our user interface, and comparison algorithms, in addition to the databases for our camera images, officials, and other users. We will also have to set up the wireless camera transmission, and also the system for observing the states of each camera. We will allocate 4 months of time in order to create the detection system.

Sam will be responsible for creating the comparison algorithms. In order to create the comparison algorithms, she will work with officials to create a rating system to analyze the state of bluffs observed, which will be done over 2 weeks. After this rating system is created, a machine learning algorithm to learn this system, and observe tells through the images given by the cameras will be created. This will take place over 2 months with appropriate testing being done to ensure the accuracy of the algorithm.

Antonio will be responsible for creating the user interface. One feature of the system is the ability to control the cameras set up along the bluffs. This will be the first task in creating the user interface and will allow users to control these cameras remotely, it will be created within the first 3 weeks of the project. This user interface will include interaction with the comparison algorithm, allowing people who use the interface to view the ratings given by the algorithm. This will be created alongside the comparison algorithm over a period of 1 month, and will be a major part of the testing phase for the algorithm. Qualified officials and maintenance workers will also have the ability to login to the central processing system, so a login system will be created over a period of 3 weeks, alongside other tasks, and will give users the ability to access the system based on their credentials that are given during an account’s creation. The interface will also send messages to relevant officials when an at risk rating has been observed, which is a rating of 4 or 5.

The relevant officials' contact information and affiliation will be included in the user/official database. Joshua will be responsible for creating the databases and setting up the cameras. Over 1 month six cameras will be set up along the bluffs, along with the necessary wireless transmission set up such as the routers. A database for the camera images will also be set up, with 2 sections one for holding newer photos(1 month or less) and also older photos(1 year or less), this task will be done over 2 weeks. In addition to the camera image database, a database including officials that could be

contacted in case of an emergency, and that will also be used to store future user login information will be created. Alongside the login system, this task will be done over a period of 3 weeks, and will be tested during the comparison algorithm testing for its ability to effectively contact relevant officials.

Test Plan

Unit Testing

- AlgorithmInteraction obj1.isBusy(boolean)
    - Test case where the algorithm interaction is not busy
        AlgInteration obj1.processingStatus = false → function should return false
        <False, isBusy()> (output, feature)

    - Test case where the algorithm interaction is busy
        processingStatus = true → function should return true
        <True, isBusy()> (output, feature)

In order to test this feature, certain preconditions must be met. In this test case, the member variable status must be manually defined. Once this precondition is met, then the actual testing can continue. There is no parameter for the feature so you run the feature with just the class object. The expected output would be true or false, depending on what the status is set too. Some possible errors would be if the output was the opposite of what was desired, meaning that the function does not work properly. It is possible that the status was not properly set, so this function could not give the desired output, but works as intended.

- Camera obj1.isEnabled(cameraID)
    - Test case where camera 1's status is enabled (true)
        Camera obj1.status = True
        Set CameraID = 1
        <True, 1, isEnabled> (output, input, feature)
    - Test case where camera 6's status is disabled (false)
        Camera obj1.status = False
        Set CameraID = 6
        <False, 6, isEnabled> (output, input, feature)

The precondition for this function is that the member variable status is set to either true or false (depending on which desired output you are looking for). The function setStatus will control the definition of the variable status, but for testing purposes, it is best to manually define the variable. Getting the camera id is also important, as that is what will be the input for the method. Testing this method involves passing the id of the camera as an argument into the method. The

desired output is the status of the camera. Some bugs would be if it gave the wrong output as well as if the id is not properly defined.

Integration Testing

- Rating getRatingFromAlg()
  //set Algorithm Interaction Alg.ImagewithRating= <qW3e1r2tyuioasdfgh, 4>
  //default value of int variable in Rating class rate.rating = 0
  rate.getRatingFromAlg() → change value of int value from 0 to 4
  rate.rating = 4

In order to test the function, getRatingFromAlg() from the Rating class, the precondition of setting the variable ImagewithRating from the Algorithm Interaction class to a <string,int> pair. After this precondition is met, with the knowledge that the default value for the rating variable is 0, the getRatingFromAlg() function can be tested. The function will separate the int value from the ImagewithRating pair, and store that value in the rating variable, fulfilling the postcondition of changing the value of the rating variable.

- setStatus(newStatus, cameraID, jobCode)
  - Test case where permission to change camera status is granted for the inputted job code
    //Set Credentials Credential.jobcode = 123456
    //Set Camera current status camera2.status = true
    //When camera class called, its called with the cameraID parameter
    Cameras camera2.setStatus(false, 2, 123456) →if jobcode has correct credentials, set camera2.status to false
    Camera2.status = false //jobcode with correct credentials
  - Test case where permission to change camera status isn't granted for the inputted job code
    //Set Credentials Credential.jobcode = 654321
    //set Camera current status camera2.status = true
    //When camera class called, its called with the cameraID parameter
    Cameras camera2.setStatus(false, 2, 654321) →if jobcode has correct credentials, set camera2.status to false
    Camera2.status = true //jobcode with incorrect credentials

The function setStatus(newStatus, cameraID, jobCode) of the Camera class requires the precondition of setting the jobcode of the employee trying to change the status of the camera in the Credentials jobCode variable, and setting the current status of the camera in the Camera class' status variable. After these preconditions are met there are two test cases for the setStatus function. In the case that the job code has the correct credentials to change the camera status, the postconditions of the test will be that the status variable will change to the newStatus that is given as a parameter in the function call. In the case that the job code doesn't

have the correct credentials to change the camera status, the postconditions of the test will be that the status variable will not change to the newStatus that is given as a parameter in the function call, and the variable will be the same as before the function call.

System Testing1

ResultingAction test1.enactAction(String linkToAction)

- Test case with linkToAction https://alertAuthorities/forCamera2/urgent executes properly
  // set ResultingAction test1.ratingCategoryAndCamID = <5,2>
  // use previously tested and verified function to set precondition
  test1.determineAction(ratingCategoryAndCamID) to return linkToAction = https://alertAuthorities/forCamera2/urgent

  test1.enactAction(linkToAction) → expected output = true

The function enactAction() from the ResultingAction class requires the parameter String linkToAction and returns a boolean value that is true if the link executes properly and false otherwise. In this test case, we set the precondition values so that we can have an exact expected output of true because the link to action is from the built-in options and we know that it works properly. In a typical scenario of the system, the linkToAction is derived from the return value of the function determineAction() also from the ResultingAction class that takes in an integer pair ratingCategoryAndCamID() parameter that is derived from setRatingCategory() and extractCamID() in the Rating class. These functions use the imageWithRating pair of String imageURL and integer rating that is accessed from the AlgorithmInteraction class through the getAlgResults() function after checking isBusy() returns true. These algorithm results were determined from the String image URL returned from the transmitImages() function in the Cameras class after checking isEnabled(). All of this occurs after a qualified individual logs in through the promptUser() and attemptLogin() functions in the Credentials class which initiates the system to enact the flow of functions described above. If the linkToAction is valid, but it does not execute the resulting actions properly, including contacting authorities and having the contact recognized, then the function would return false, indicating an issue with the execution of the link on the enactAction function.

- Test case with linkToAction being https://youtu.be/dQw4w9WgXcQ (Rick Roll video) does not execute properly
  // manually set ResultingAction test2.linkToAction = https://youtu.be/dQw4w9WgXcQ

  test2.enactAction(linkToAction) → expected output = false

In this test case, we set the parameter linkToAction to be a youtube link, instead of one of the valid links to a resulting action based on the systems distinction of an issue, so therefore no feedback that the action had been properly enacted will occur and the function should return false. If the function returns true, then there is an issue with the determination of boolean return

value within the enactAction() function. A normal use case flow for actual scenarios involving the use of this function is described in the above test case description.

ResultingAction test1.setCameraStatus(boolean status, int camID)

- Test case with parameters status = false and camID = 3 which sets camera 3 to be no longer active
  // set Credentials user1.jobCode = 123456 which is established as valid for changing camera status
  // set camera with camID = 3 to have status = true
  // Camera cam3.camID = 3, then cam3.status = true

  ResultingAction test1.setCameraStatus(false,3) → expected result is that the camera with camID = 3 now has status = false

The function setCameraStatus() from the ResultingAction class requires the parameters of a boolean status and an integer of camID. While there is no return type to this function, the result is checked by accessing the member variables of the Camera object on which the function is called. We initialize the camera ID to 3 and the status to true so that we start out knowing we are working with the third camera and it is enabled. Then, when we run the setCameraStatus() function, we can recheck the status of the camera with camID = 3 which should now be false if the function worked properly, which we can do using the previously verified isEnabled(camID) function that will return the boolean status of the camera the the specified ID. In a real use of the setCameraStatus() function from the ResultingAction class, it would have a resulting call to the setStatus() function in the Cameras class that takes in parameters of newStatus and camID from the original function call, then it also takes in a jobCode that is accessed from the Credentials class which acquired the jobCode when the user logged in, which in this case is 123456 for one of the verified maintenance personnel. Then, the setStatus() function in the Cameras class uses the return value from the givePermissions(jobCode) function to determine whether the specified change in camera status is a permission granted to that user. If it is a qualified user, then the status member variable of the camera with the specified ID will be changed to the value of the newStatus boolean, which in this test case is false. Reasons there could be an error in the result of the function is if an invalid jobCode is provided or if there is a problem with changing the status attached to the specified camera object. Other test cases for this function could include changing the status from false to true, verifying that an a user with an incorrect jobCode cannot enact changes, and ensuring that setting a camera to true that is already set to true does not change its status.