Team 1 - VIR - Atlas

Visible Infrared Atlas

NASA New Technology Report Number - 1615310307

**Final Report**

# **Project Overview**

Our team’s original objective was to build an accurate visible and NIR (near-Infrared light) spectrum (or Color-Infrared) mapping software specifically for STELLA (Science and Technology Education for Land/Life Assessment) that will cartographically include and/or display STELLA’s other sensor readings in a user friendly and visually appealing GUI (Graphical User Interface). Because of our clientele, our objectives have been changed to also include accurately calculating and displaying various Vegetation Index maps and two temperature maps in VIR-Atlas. We’ve also included the ability to let users create annotations, save, and re-upload files, for convenient research and education purposes.

## **Scope and Objectives**

Our team has been presented with the unique opportunity to work with a newly developed and nearly patented piece of hardware called STELLA, a handheld device developed by NASA researchers that parallels the technology used by Landsat. Its purpose is to easily survey and record numerous environmental conditions for a given geographic area (relatively small) with an accuracy comparable to Landsat. STELLA’s various sensors measure conditions such as humidity, air pressure, and surface temperature. However, our team is most interested in its visual and near-infrared light spectrum readings.

The data that STELLA generates presents similar applications and opportunities that the Landsat program provides. This data is fundamental in many fields like ecological analysis, preventative forest fire management, search and rescue surveys, land-use, and development planning, and so much more. The best way to utilize and understand the data for almost any of these applications is to visualize it into a map, a feature STELLA isn’t currently capable of. The maps we have calculated and produced for research and education development include the following: Visual Light Spectrum, Infrared Light Spectrum, Temperature, Air Temperature vs Surface Temperature, Normalized Difference Vegetation Index, Enhanced Vegetation Index, Soil Adjusted Vegetation Index, and Modified Solid Adjusted Vegetation Index.

Paul Mirel, the primary developer of STELLA, highly advocates for educational practices that he hopes can be used from STELLA. With this and research applications considered, our software also includes annotations to be added, edited, and deleted to the maps displayed, so that users can make note of specific points in the area and re-reference their notes. We’ve also implemented a satellite image display and user image display next to the map, so that users can more easily compare the maps VIR-Atlas generates to an actual image of the area being covered.

## **Supplementary Requirements**

* + 1. **Assumptions**

The user will use a modern OS on a pc (works best on Windows for the time being). The user will also have obtained the necessary data files in the correct format when taking raw data. This will require a drone that creates a CSV file containing timestamps and GPS coordinates and a STELLA device.

* + 1. **System Expectations**
       1. Extensibility

The software will be developed in a modular manner, which will allow the addition of new features without requiring rewriting files.

* + - 1. Reusability

The software will have generic functionality which can be reused across subsystems.

* + - 1. Reliability

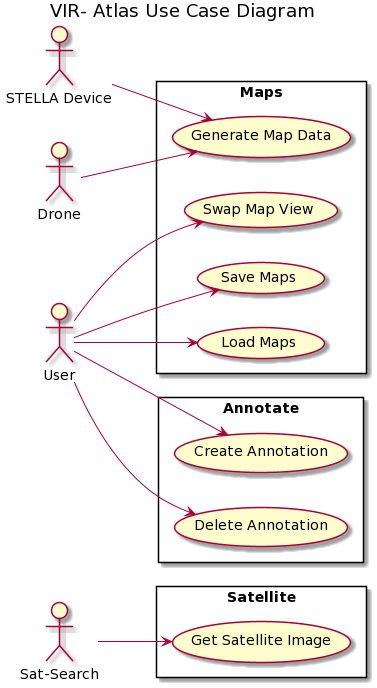
The software will perform desired tasks adequately.

* + - 1. Documentation

Every class and function will have extensive comments detailing functionality and necessary information. A document describing software functionality, proper use, and vital installations will be included with the software.

# **Customer Requirements**

## Use-Case Diagram



## Actor Description

* + 1. User: The actor ‘User’ is anyone who uses the VIR-Atlas system to view data generated by a STELLA device. The User can view a map, annotate it, and save their map and annotations. A user can load a previous annotated map to view their notes.
    2. Both the Drone and STELLA actors produce data files that VIR-Atlas will interpret to display a map. These files are selected by the User when they opt to display a map.
    3. The Sat-Search actor produces the Satellite image of the area based on the latitude and longitude values from the STELLA device and Drone input.

## Use Cases

* + 1. Generate Map Data:
  1. Brief Description:
     1. This allows users to enter data from both a STELLA device and a drone flight. The user can view the area’s 8 available maps alongside a satellite image of the area they collected data for.
  2. Flow of Events
     1. Basic Flow
        1. The User will upload the data they collected from their STELLA device to our software.
        2. The User will upload the data they collected from the drone about the flight to the software.
        3. Both data files are parsed by the software and displayed in the GUI.
        4. A satellite image will be displayed alongside the generated map.
        5. The user can then pan around the map, zoom in, and zoom out.
     2. Alternative Flows
        1. If the data isn’t recognized as an area map, the system will display an error message and ask the user to input a recognized data pattern..
     3. Special Requirements
        1. None
     4. Pre-Conditions
        1. The user must have data from a STELLA device and a drone flight.
     5. Post-Conditions
        1. The user can then open another data file or save the current one.
     6. Extension Points
        1. None
     7. Swap Map View:
  3. Brief Description:
     1. This use case allows the user to look at one of the 8 different map options that VIR-Atlas can display. These maps include: Visual Light Spectrum, Infrared Light Spectrum, Temperature, Air Temperature vs Surface Temperature, Normalized Difference Vegetation Index, Enhanced Vegetation Index, Soil Adjusted Vegetation Index, and Modified Solid Adjusted Vegetation Index.
  4. Flow of Events
     1. Basic Flow
        1. The user is viewing the default display map (Temperature map) after uploading data files
        2. The user can select “View” in the menu bar, where every map display possible will be an item in the “View” menu.
        3. User selects the map they would like to view
        4. Map displays in main GUI
        5. The user can repeat this process as needed.
     2. Alternative Flows
        1. None
     3. Special Requirements
        1. None
     4. Pre-Conditions
        1. The user must have uploaded data from a STELLA device and a drone flight, to be processed by map generation
     5. Post-Conditions
        1. None
     6. Extension Points
        1. None
     7. Save Maps
  5. Brief Description:
     1. The user can save a map, with annotations included, as a special map file type we’ve created to be compatible with VIR-Atlas
  6. Flow of Events
     1. Basic Flow
        1. The user is viewing one of the 8 possible displayed map for the data files they’ve uploaded (with annotations or without)
        2. The user selects “File” in the top menu bar
        3. The user selects “Save File” in the menu bar options
        4. .vmap File of the saved map is downloaded onto the users OS
     2. Alternative Flows
        1. None
     3. Special Requirements
        1. None
     4. Pre-Conditions
        1. The user must have uploaded data from a STELLA device and a drone flight, to be processed by map generation
     5. Post-Conditions
        1. None
     6. Extension Points
        1. None
     7. Load Maps
  7. Brief Description:
     1. The user can open a previously saved .vmap file, and view that map in VIR-ATLAS
  8. Flow of Events
     1. Basic Flow
        1. User starts up VIR-Atlas (or is already viewing something)
        2. In the top menu bar, the user selects “File”
        3. User selects “Load Previous Map”
        4. User selects a valid .vmap file to load into VIR-Atlas
        5. VIR-Atlas displays the map data loaded
     2. Alternative Flows
        1. If the user does not select a .vmap file, or the file they previously saved is not valid, an error message will appear.
     3. Special Requirements
        1. None
     4. Pre-Conditions
        1. The user must have already uploaded STELLA and the drone data, and have saved that map file as a .vmap.
     5. Post-Conditions
        1. None
     6. Extension Points
        1. None
     7. Create Annotation:
  9. Brief Description
     1. This allows the user to mark areas of the map, enter notes about them, and save their annotations.
  10. Flow of Events
      1. Basic Flow
         1. The user will select an area or point on the map.
         2. An attribute screen will pop up, displaying the point for that data, or letting the user know that there is no data recorded at that point.
         3. Attribute window has a text box for users to enter annotations
         4. User can either press the “Cancel” button if they would not like to save that annotation, or the “Save” button if they would like to save the annotation to the map
         5. Once an annotation is saved, its annotation object will appear on the main GUI’s annotation box, as will a pin button on the point an annotation was made on.
         6. The user can repeat this process as needed.
      2. Alternative Flows
         1. Users can select an annotation in the annotation box of the main GUI and select the “Edit” button to open that annotation. The same process can be done by simply clicking an annotation button saved on the map.
         2. An attribute screen will pop up, displaying the point for that data, or letting the user know that there is no data recorded at that point.
         3. Attribute window has a text box for users to enter annotations, and includes the old annotation information entered previously
         4. User can either press the “Cancel” button if they would not like to save that annotation, or the “Save” button if they would like to save the new annotation to the map
         5. Map and annotation box will delete the old annotation made and replace it with both with the information from the new annotation that the user has saved
         6. The user can repeat this process as needed.
      3. Special Requirements
         1. None
      4. Pre-Conditions
         1. The user must have one of the 8 maps displayed already.
      5. Post-Conditions
         1. The user can open a new map and enter more annotations or save the current one.
      6. Extension Points
         1. None
      7. Delete Annotation:
  11. Brief Description:
      1. This allows the user to delete any annotation saved for a map.
  12. Flow of Events
      1. Basic Flow
         1. The user will select a previously saved annotation in the annotation box of the main GUI
         2. Once an annotation is selected, the user can press the “Delete” button to remove this specific annotation from the map
         3. The annotation pin, and object will be removed from the map and the output file.
         4. The user can repeat this process as needed.
      2. Alternative Flows
         1. None.
      3. Special Requirements
         1. None
      4. Pre-Conditions
         1. The user must have one of the 8 maps displayed already
         2. User must have a saved annotation on the map already
      5. Post-Conditions
         1. None
      6. Extension Points
         1. None
      7. Get Satellite Image
  13. Brief Description:
      1. Automatically retrieves a satellite image based on the data provided by the user
  14. Flow of Events
      1. Basic Flow
         1. The largest and smallest coordinate tuples are gathered from the provided data to determine the upper and lower corners of the image range.
         2. The SatSearch package retrieves an image corresponding to the area generated by these corners.
         3. The image is displayed in the GUI
      2. Alternative Flows
         1. The user can view any image they provide in place of the automatic satellite image.
      3. Special Requirements
         1. Though a satellite image will always display given GPS data, a large coverage area is desirable to easily compare the satellite image to the generated map.
      4. Pre-Conditions
         1. The user must load GPS data and STELLA data
      5. Post-Conditions
         1. None
      6. Extension Points
         1. None

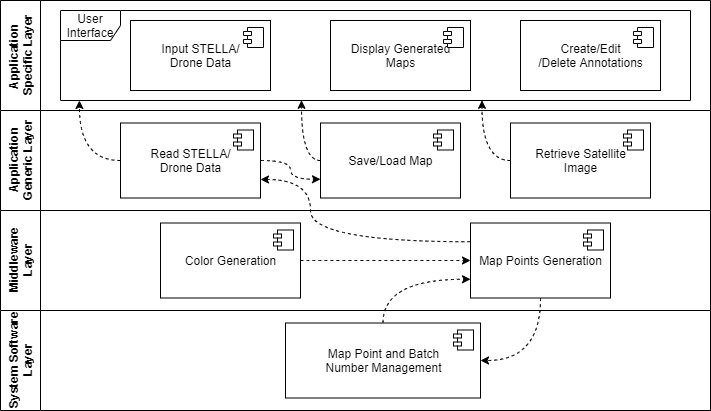
# **Architectural Design**

VIR-Atlas follows a Pipe and Filter architectural style; that is, the software is dependent on the input data the user provides from STELLA and the drone’s GPS coordinates corresponding to its flight path. These data source components are then consumed by the system sync components (i.e., backend subsystems) in order to output the 8 generated maps that VIR-Atlas calculates.

An alternative system architecture style considered for the system was Client-Server based. Essentially we considered having the client be the interface for the user to input their drone path data and STELLA stella data, and the servers be various mapping generation software we’ve created from the backend and linking to Google Maps to get a standard map of the area. However, because of time constraints, we decided that NOT generating the maps over a server would be more reasonable for getting this software done in the time we had. We also could not link up to Google Maps because it has a very expensive API key you must buy monthly. So we implemented a python package “sat-search” to display the satellite image of the area we wanted.

## Subsystem Architecture

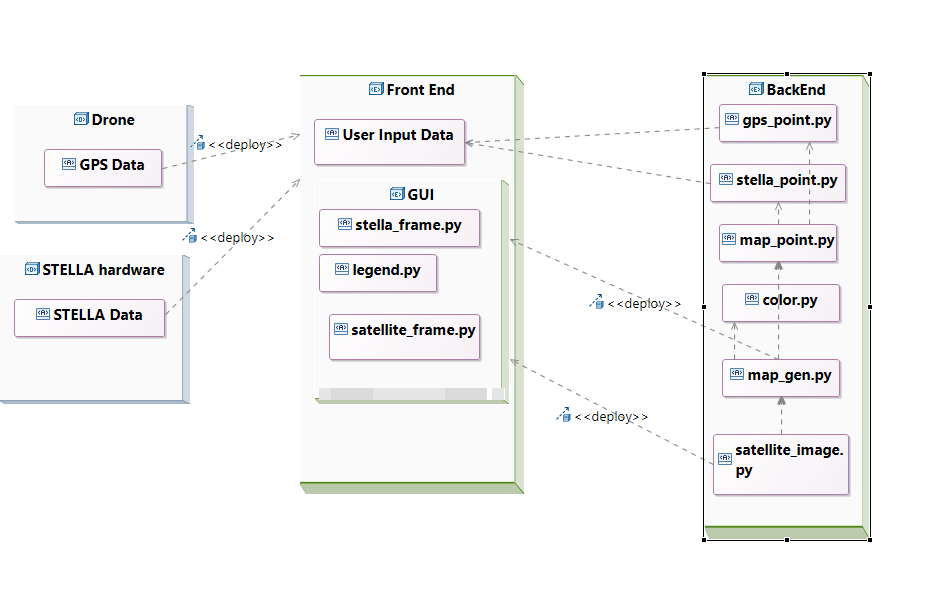
Below is the dependency diagram correlating to VIR-Atlas’s subsystem architecture based on the four different architectural layers: the Application Specific Layer, the Application Generic Layer, the Middleware Layer, and the System Software Layer. We found that this architecture style was most effective for our application because its accuracy and functionality are dependent on user input, represented in the Application Specific Layer.



* + 1. VIR-Atlas’ Application Specific layer holds our entire user interface, including inputting data from STELLA and the drone, displaying maps of the data input, and allowing the user to make, edit, and delete annotations. These aspects of the user interface are specific only to our application, as the specific data input from the user is absolutely essential to the functionality of our map generation and is, therefore, an interface that is not reusable across applications.
    2. In the Application Generic layer, we have the processing or reading of the user input data, the saving and loading of map files, and retrieving a satellite image of the data being displayed. These components will extract the needed map points and the corresponding data from said points to acquire the necessary information to display in the GUI, which is more general to our application but still heavily dependent on the Application Specific layer, making them still somewhat Application Specific.
    3. We have the actual calculations and generation of color and map points in the Middleware layer based on the user input. These components are essential in maintaining communication between most of the software components to produce our final result.
    4. Finally, in the System Software layer, we have our management system for organizing the useful mapping coordinates and their corresponding batch numbers for possible future reference. This has a very technical and low-level functionality in our system.

## **Deployment Model**

Below is the deployment for VIR-Atlas’ system architecture. We have two devices, the drone, and the STELLA hardware, that each output data files into the front-end execution environment. This environment is associated with the BackEnd execution environment, responsible for calculations and map generation. map\_gen.py and satellite\_image.py deploy back into the front end to be displayed for the user. All map\_gen.py data is dependent on the user input data. With the specific requirements needed for this project, organizing our system this way makes the most sense

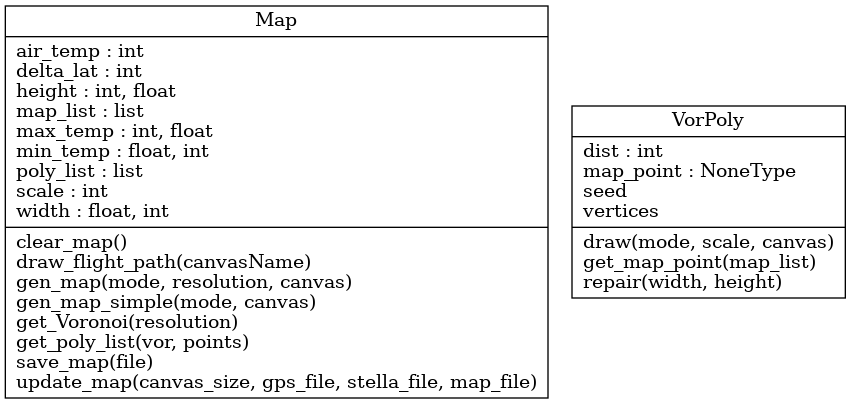
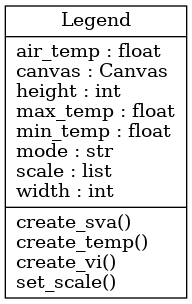
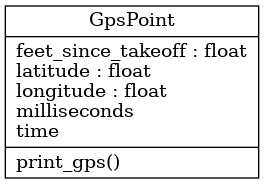
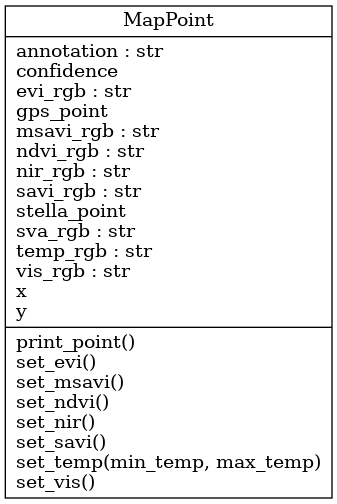
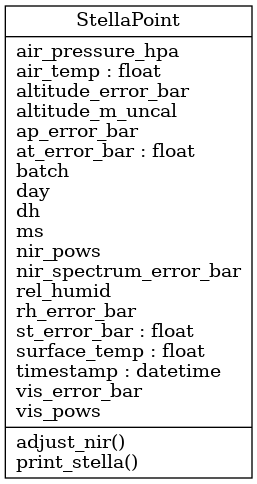
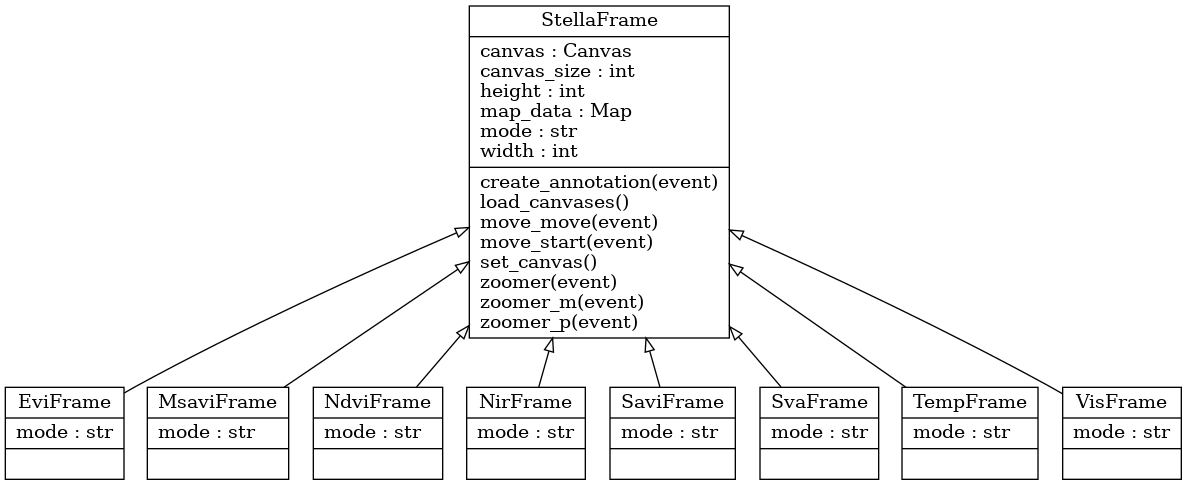
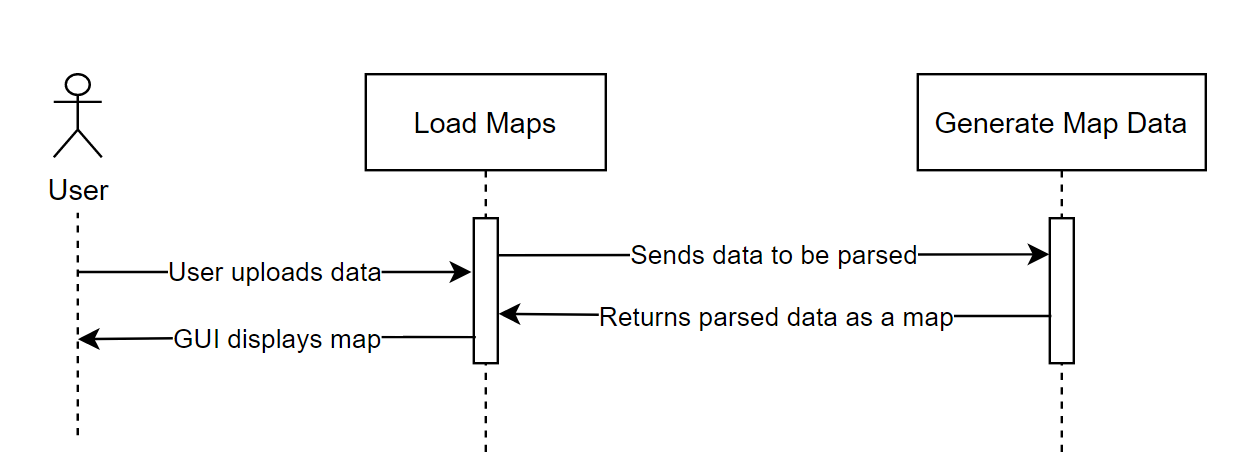


# **Use Case Realization Design**

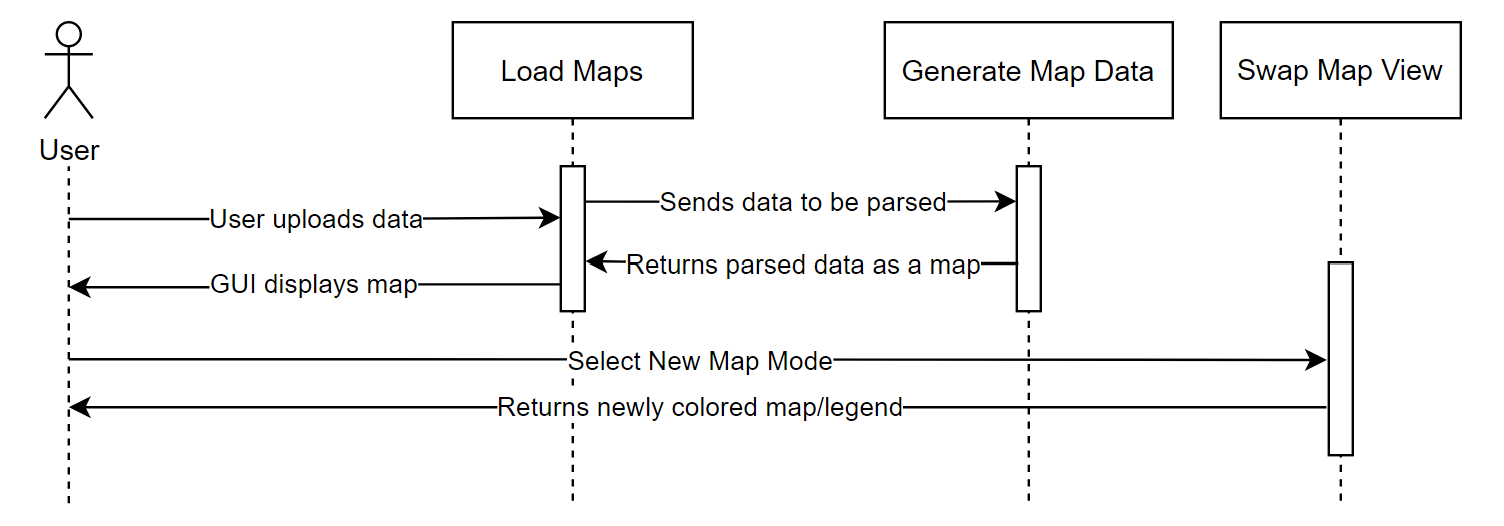
## 

Since the root object maintains the main window for the application, it's a participant in every use case. Similarly, the MenuBar initiates many interactions from the root window.

## Generate Map Data

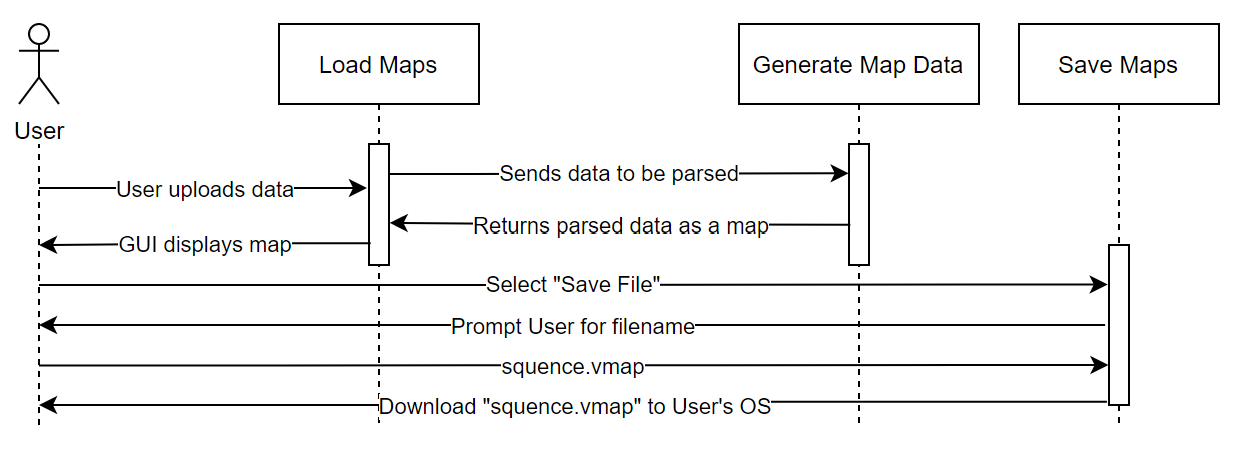


## Swap Map View



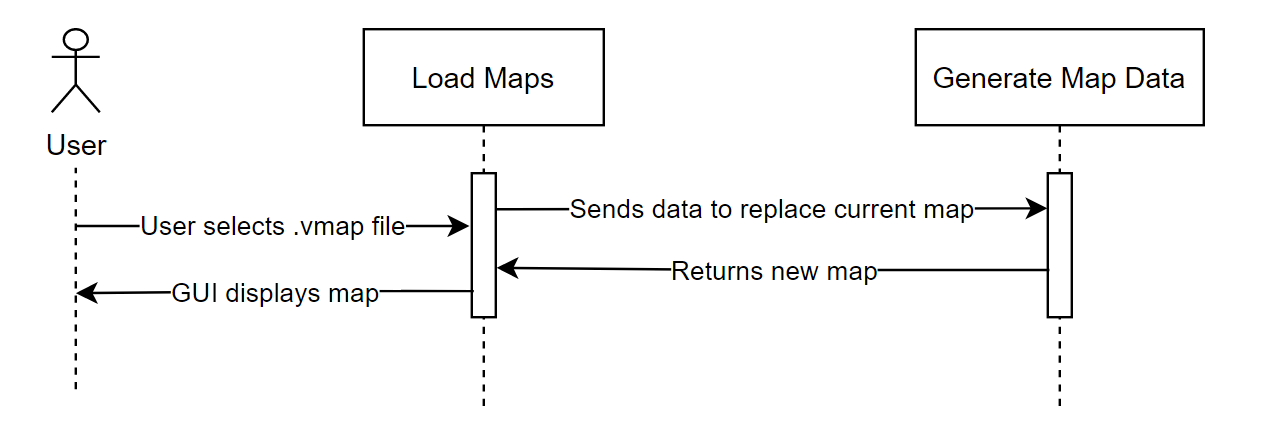
Swap Map View involves Root, and StellaFrame shown above.

## Save Maps



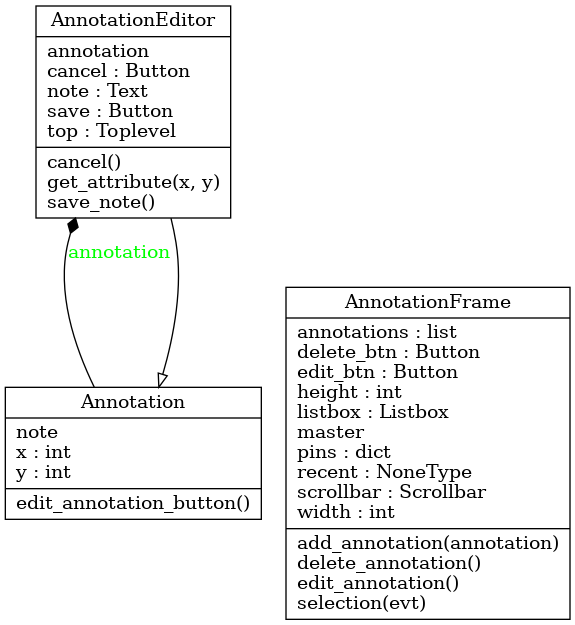
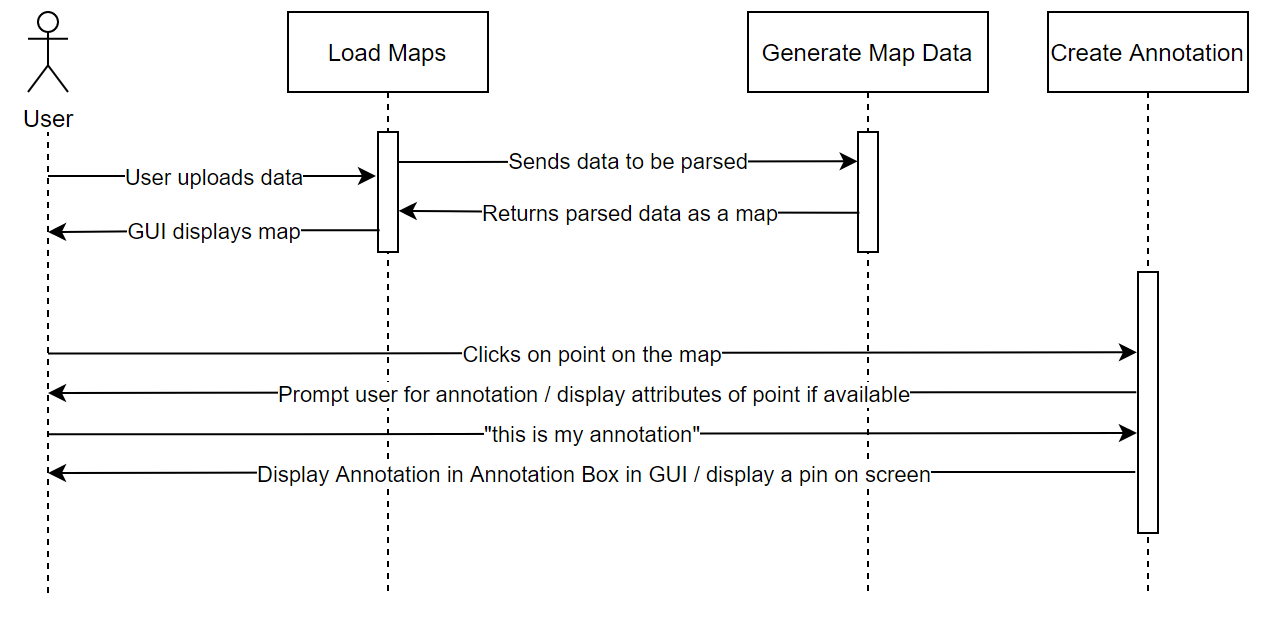
Saving maps involves MenuBar as shown above

## Load Maps

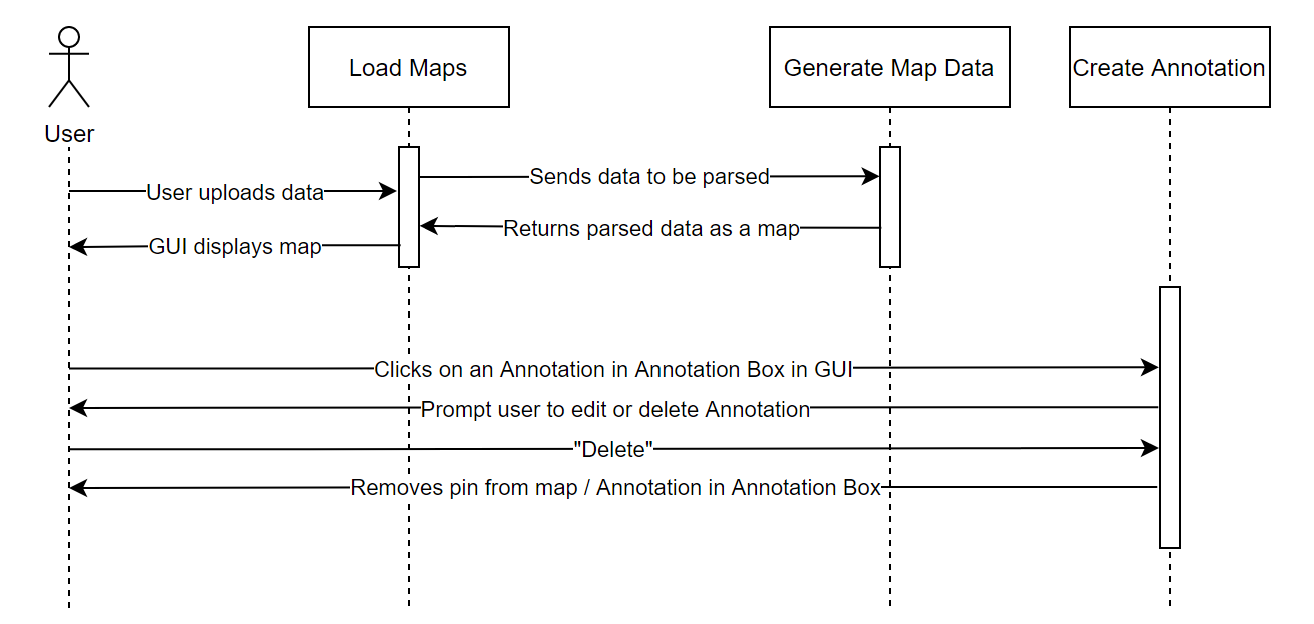


Map loading involves MenuBar as shown above.

## Create Annotation

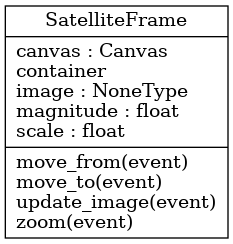
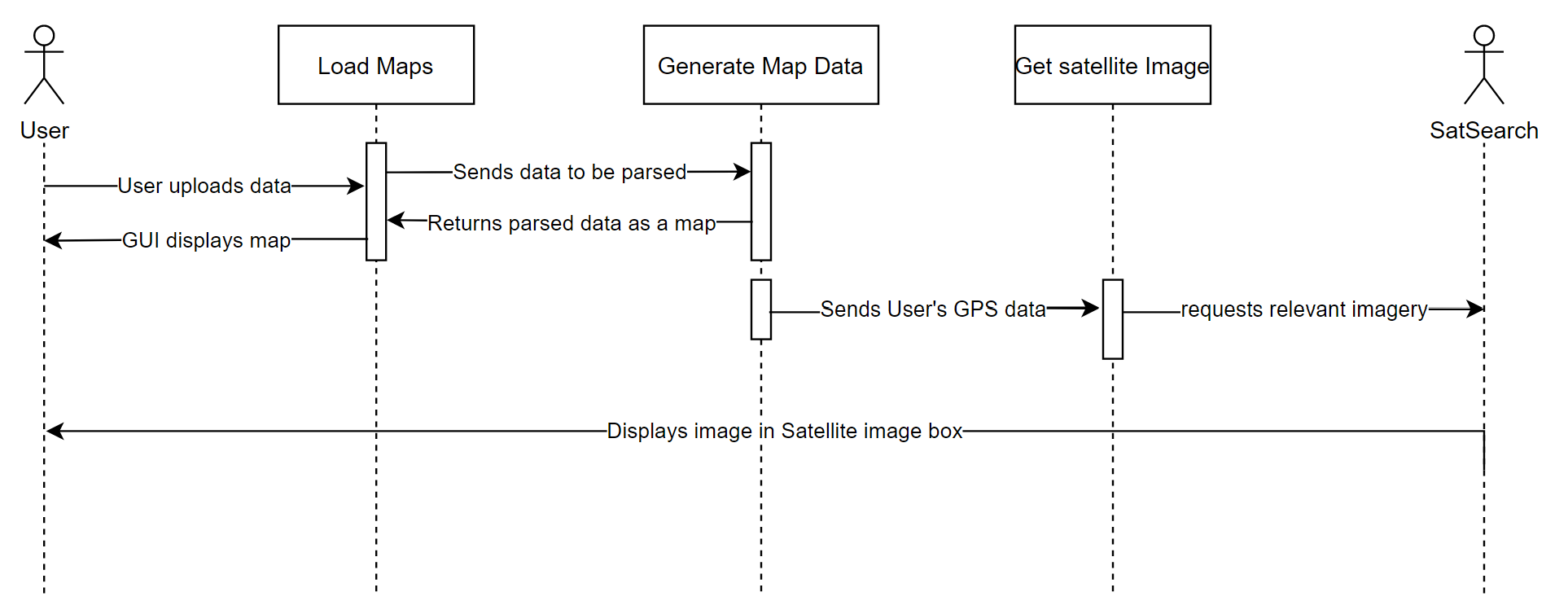


## Delete Annotation



Deleting annotations involves the AnnotationFrame as shown above.

## Get Satellite Image



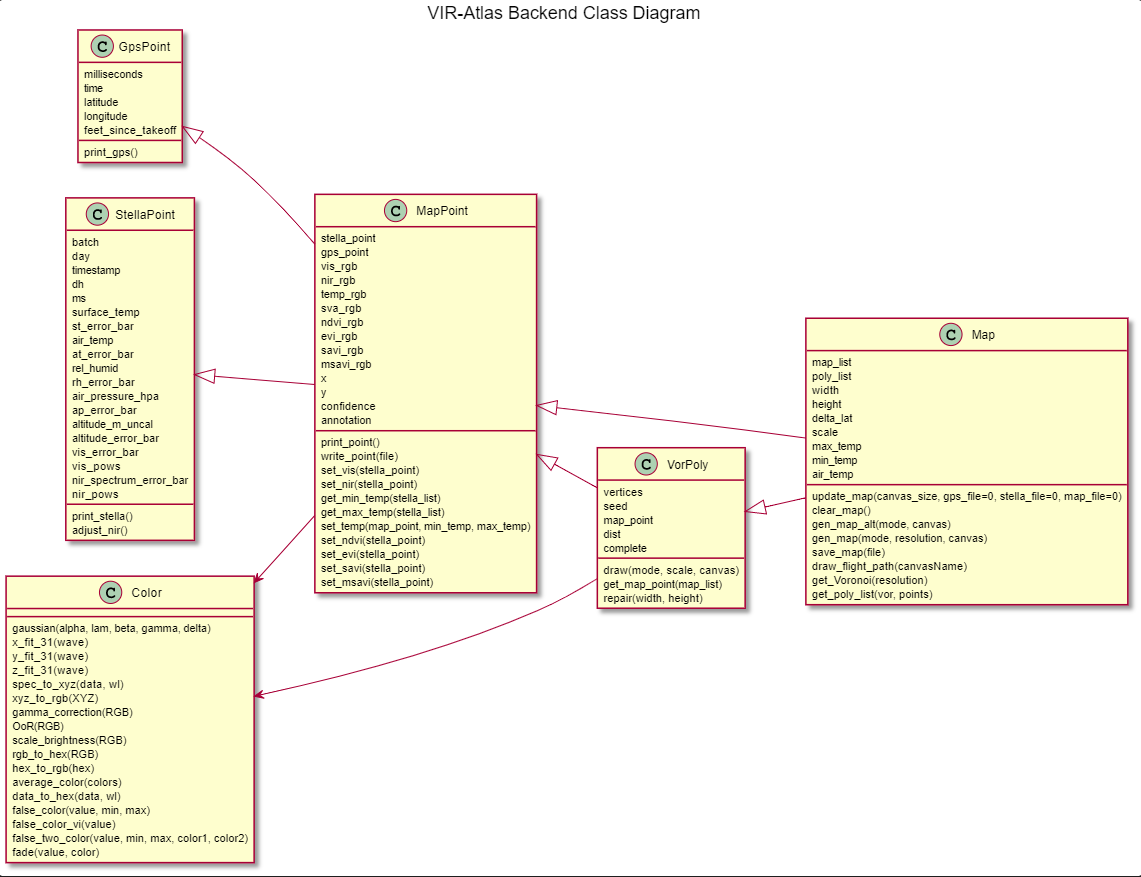
# **Subsystem Design**

VIR-Atlas is broken into two primary parts, the backend, and the frontend. The frontend displays options and deals directly with human interaction for file selection, modification, and displaying data. The backend takes the chosen file(s) and generates a map for the front end to display.

## **Subsystem A -** Backend

The backend’s primary purpose is to make meaning from two data files, one corresponding with the drone’s GPS location, and the other color and temperature data taken by STELLA. The backend pairs data points together to put STELLA data on a map based on the corresponding GPS coordinates. Once paired, polygons are generated from seeds that will be colored using the data points. The paired data points can also be saved and used to generate the same map again later without having to process and repair the data.

The UML diagram for this subsystem is below:



## Color

The Color class contains utility functions for translating data to color. This includes spectral data using CIE 1931 standards, Temperature data represented in false color, and Vegetation Index, which is calculated using equations from landsat and represented in false color.

## GPS Point

The gpsPoint class holds four pieces of data, a timestamp converted into datetime, milliseconds since takeoff, and the decimal longitude and latitude coordinates. The class itself only has one function, and that is to print itself. The rest of the file has functions that take a data file in and read the appropriate columns into a list of gpsPoints. Every data point is considered, as data files from the drone each only correspond to one flight.

## STELLA Point

Structured very similarly to GPS Point, the stellaPoint class holds all relevant data, including the timestamp, milliseconds from the beginning of the batch irradiance, temperature, and weather data. Within the class, there is a print function. Outside the class, there are functions to read in data from the STELLA data file. The primary difference here is that STELLA files are compounded with a batch code to differentiate data sets. All points are read in, but the desired batch is pulled using get\_batch().

## Map Point

The mapPoint class holds its corresponding mapPoint and stellaPoint, the converted colors to display (visual, infrared, and temperature), and the converted xy-coordinates to be placed on a Tkinter canvas. All of the functions outside of the class work to pair gpsPoints and stellaPoints based on the milliseconds from the first data point. Within this file is also the conversion process from decimal latitude and longitude to xy-coordinates.

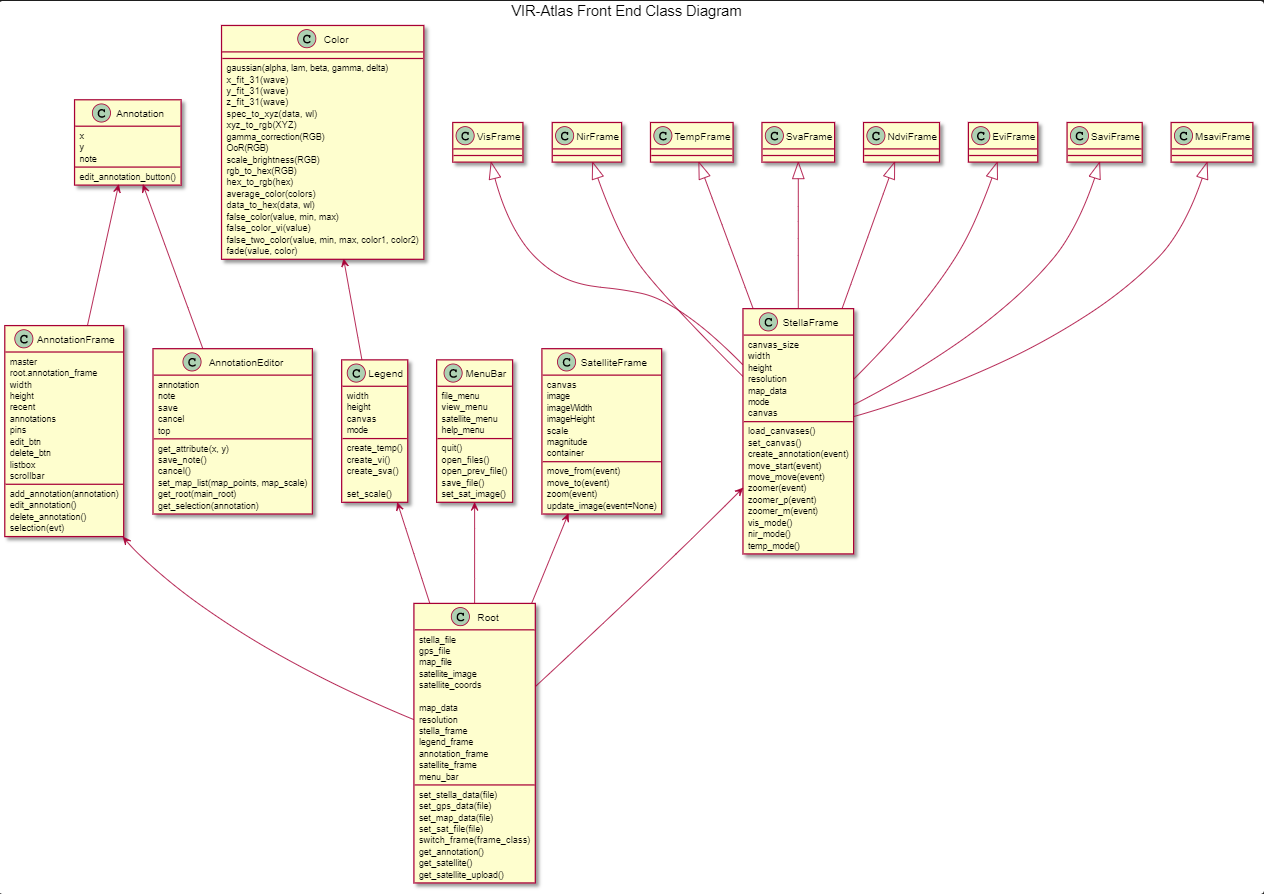
## Voronoi Polygon

A list of these Polygons are generated by Map. They are defined by the seed point used to generate them, their vertices, and the closest Map point to the seed. This allows for the color to be redefined from the Map point every time the map is swapped, rather than recalculating each polygon. Voronoi does have a flaw of creating “infinite polygons” where one or more vertices is beyond the border of the area calculated. These vertices are deleted. When the polygons are complete and have a map\_point, they can draw their area and color. The color is modified by the distance from their corresponding Map point, fading out to gray the farther away the point is from the seed.

## Map

A map is primarily defined by the list of data points and the list of polygons used to draw the map. However it also holds a few defining features of the data, such as the GPS area covered and the maximum and minimum temperature recorded for that data set. The list of Voronoi polygons is generated using a list of seeds composed of quasi-random points and the data points taken, then fortune’s algorithm is applied to the list. This provides a list of polygons that the Map can then loop through to draw.

## **Subsystem B** - Frontend



## Root

The root class sets up the main window of the GUI. It sets up the layout of all the required frames.

It is responsible for switching the frames for the STELLA frame and setting the satellite image.

## MenuBar

This class is responsible for the menu bar displayed at the top of the application screen. It offers the user access to switching maps, saving/loading/opening files, and creating annotations.

## SatelliteFrame

This class sets up a frame to display an image file. It also sets up functionality for scrolling/panning/zooming for an image. Separating the satellite image retrieval and the frame setup for it allows the user to also add a custom image to the satellite frame.

## StellaFrame

This class is responsible for displaying the STELLA canvas. It sets up functionality for displaying different STELLA canvases as well as functionality for scrolling/panning/zooming.

## Annotation

This is an object that contains all relevant data to an Annotation. An Annotation object can contain all of the point attributes (if available) and any user input on that point. If the user selects an area that is not close enough to a point on the map, the attributes will not be generated or available to the user because they do not exist. When the user initially creates an Annotation, the application will automatically find the nearest point corresponding to the area the user clicked on.

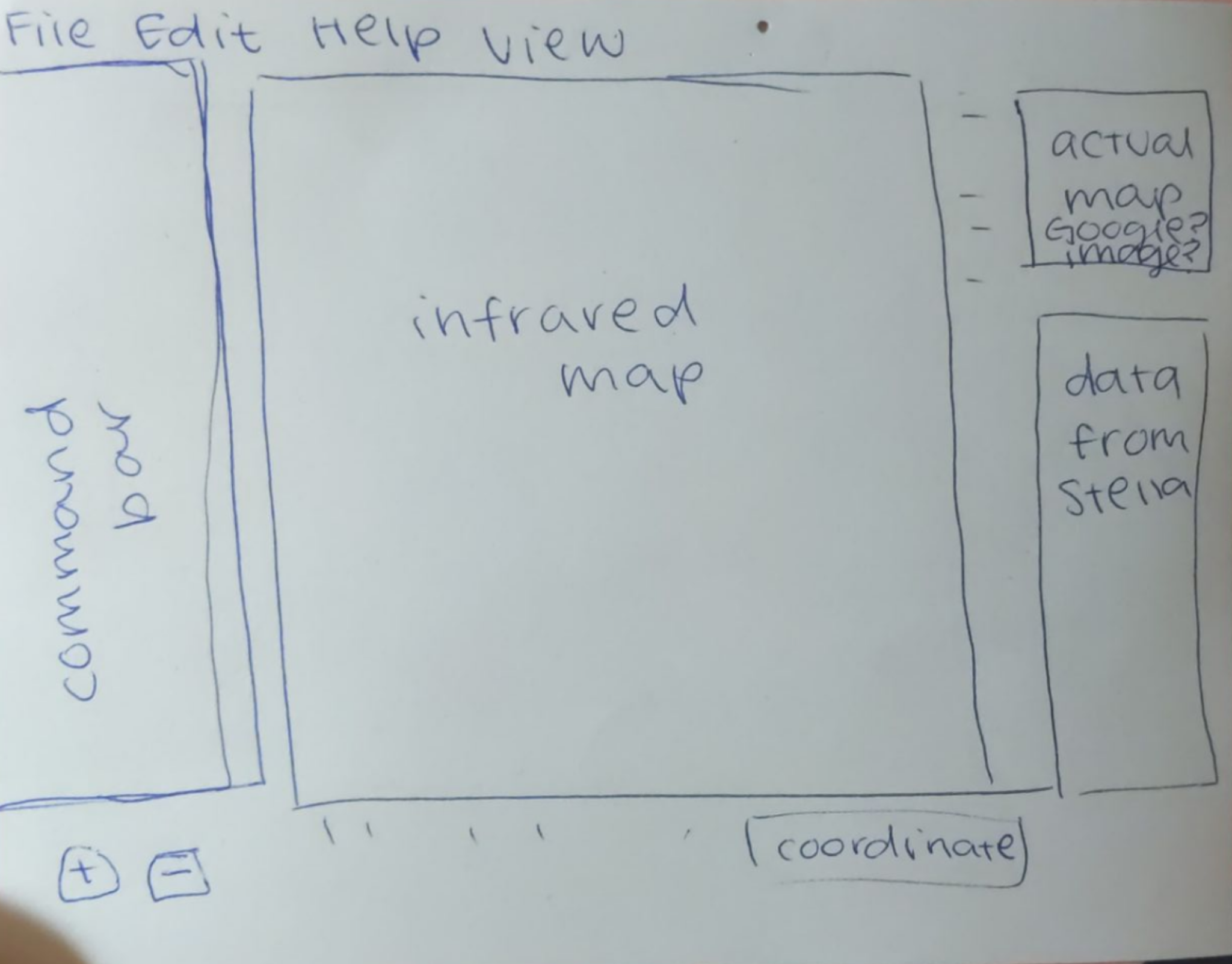
## AnnotationEditor

The Annotation Editor displays the pop-up window for editing/creating/deleting an Annotation. It also displays the aforementioned attributes attached to the Annotation object.

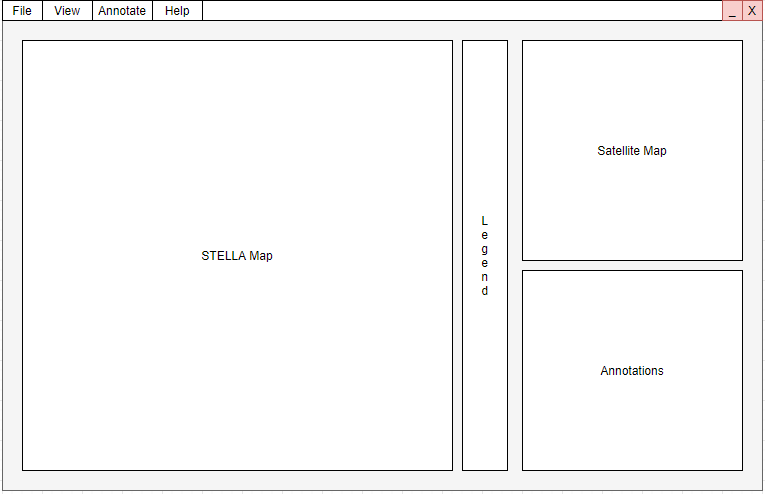
## Annotation Frame

The Annotation Frame displays in the main window of the application. It only displays a list of annotations created.

# **Human Interfaces**



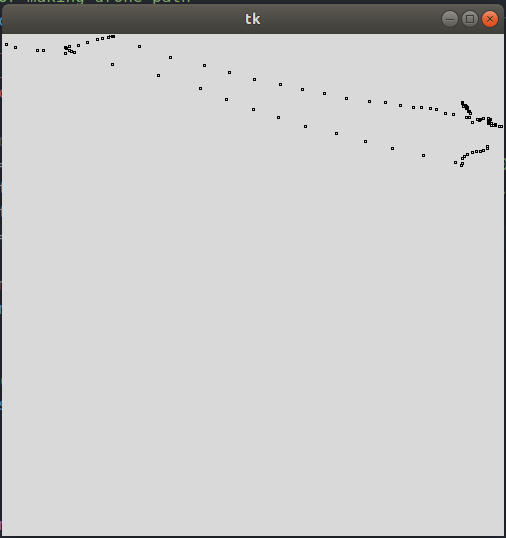
Initial Drawing Mock-Up of GUI



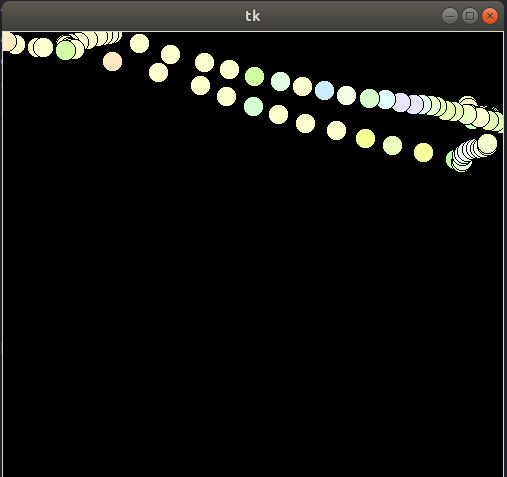
Final Mock Up Drawing of GUI



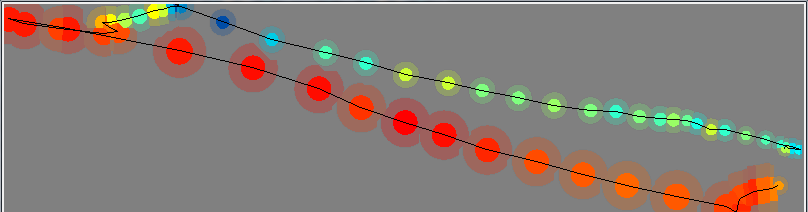
Satellite Image of First Drone Flight Data Set



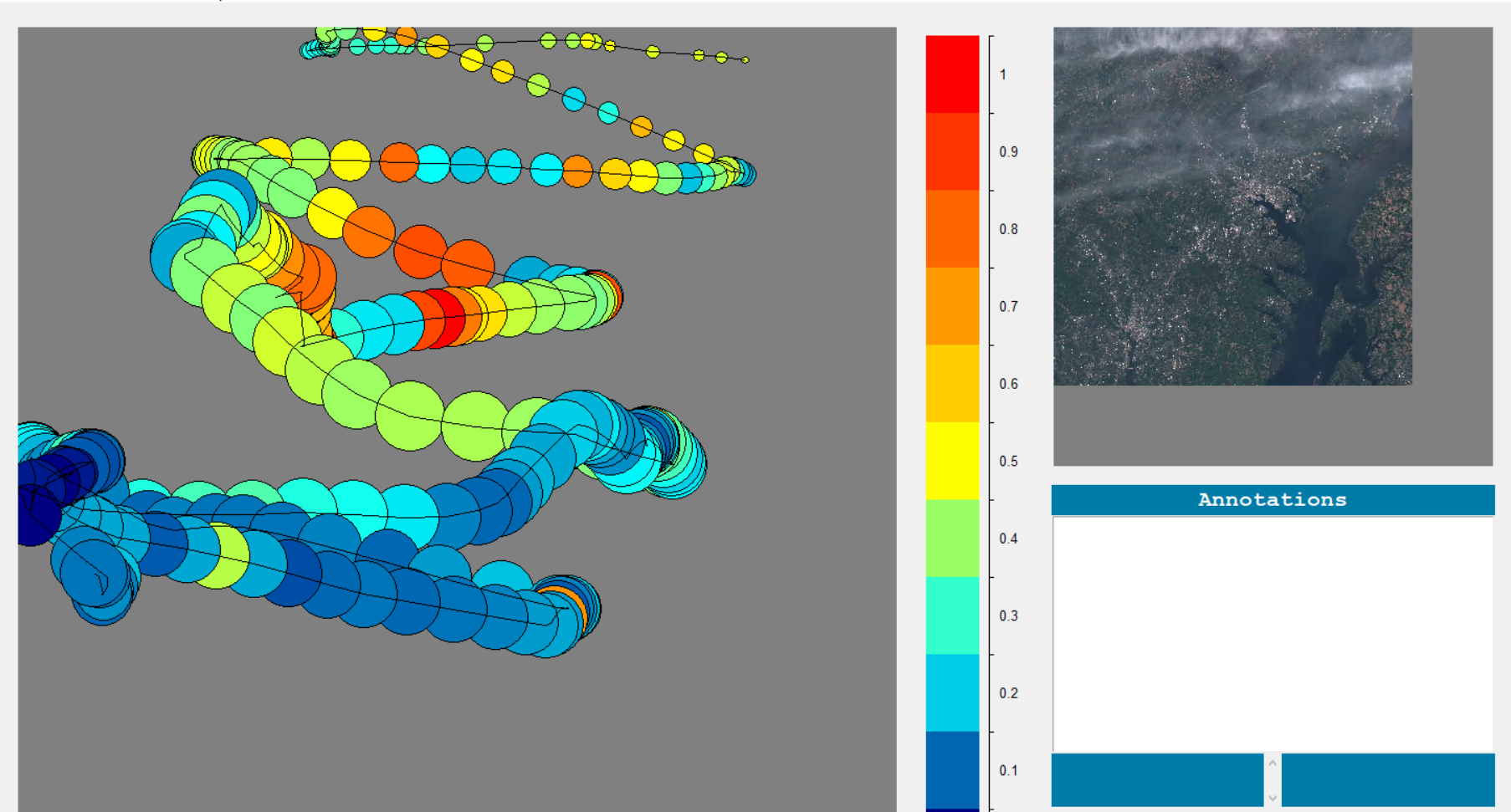
Initial Mapping of First Drone Flight



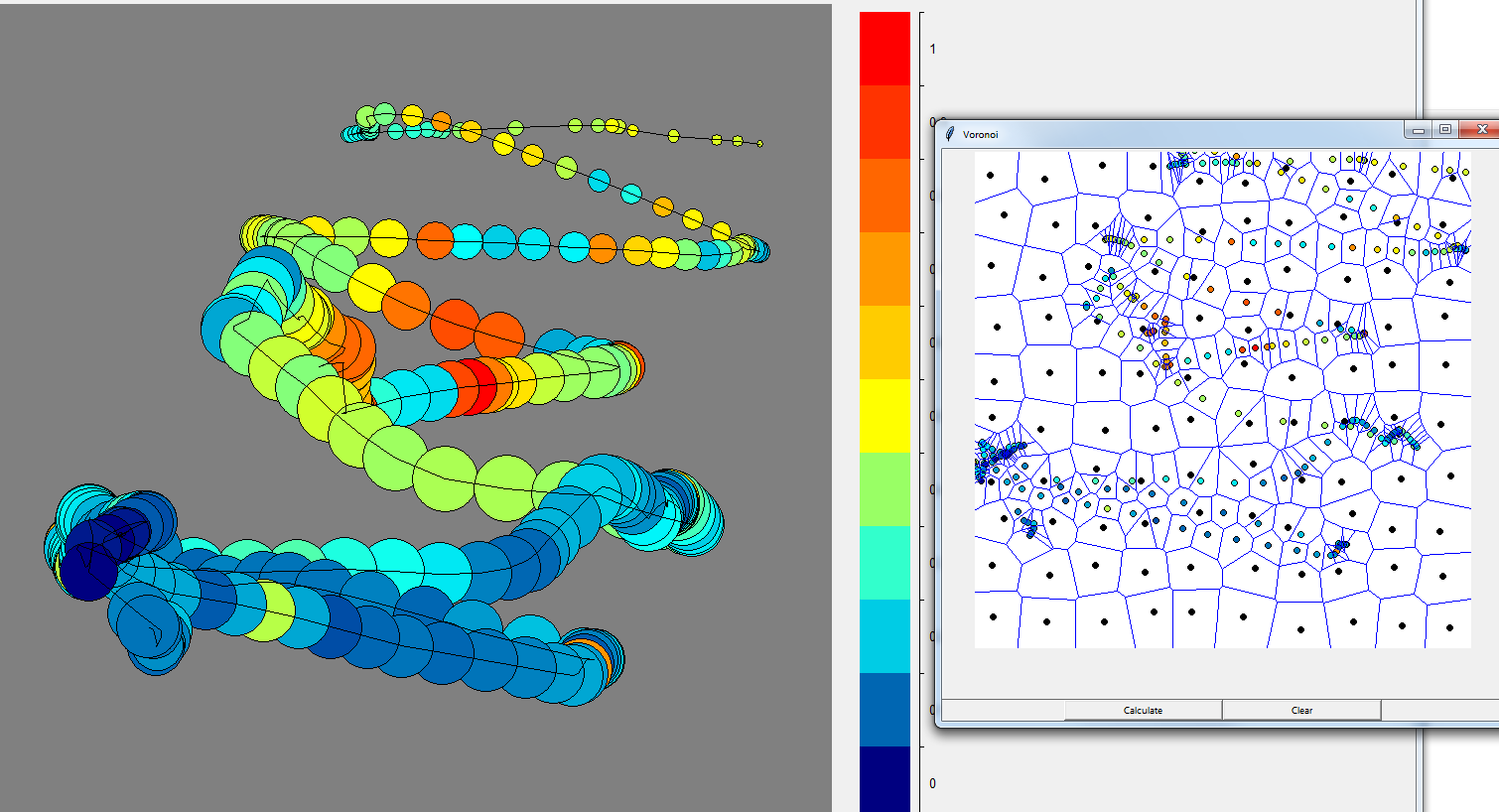
First Mapping with STELLA Data with Drone Flight Data



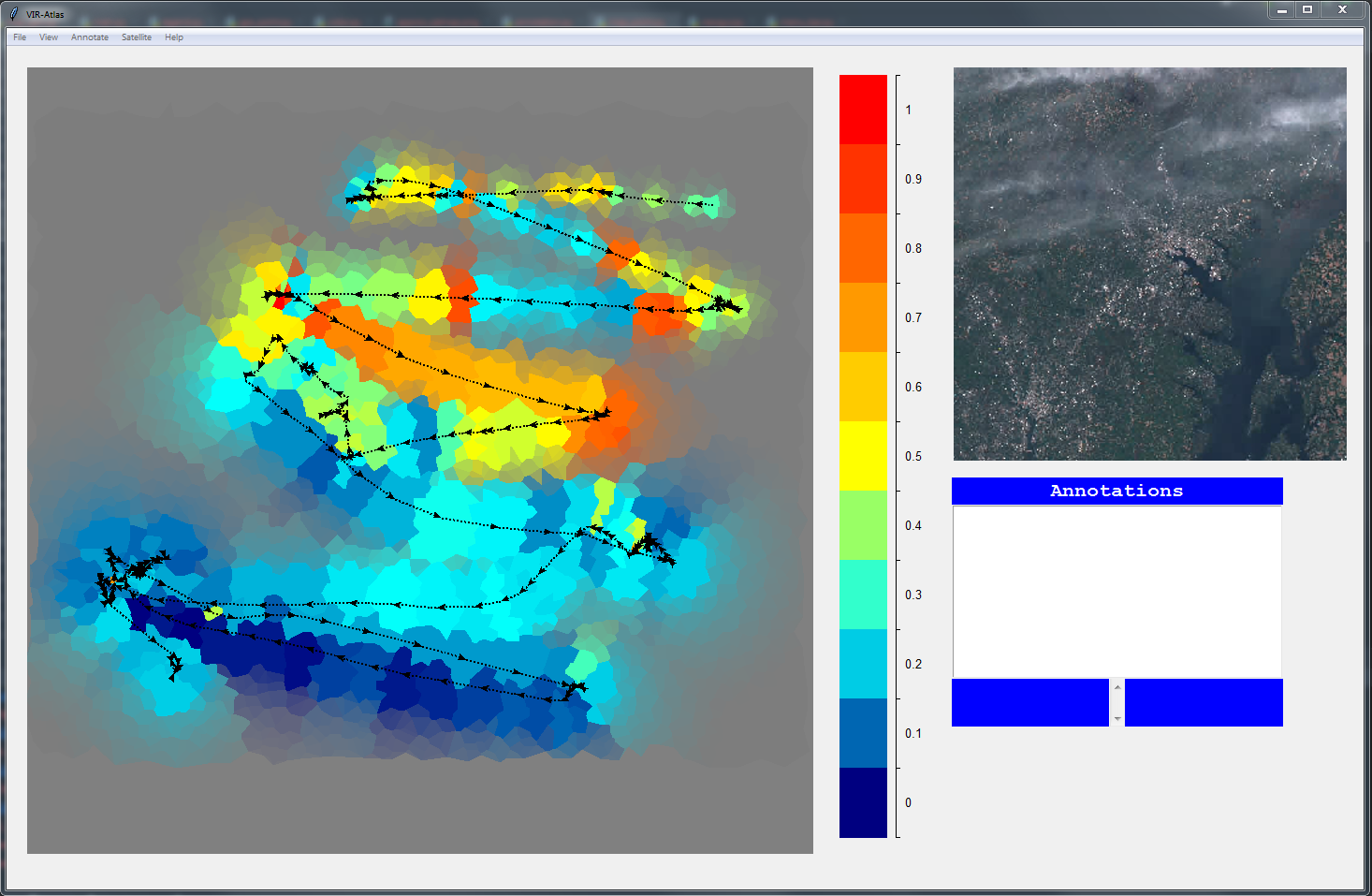
Map Generation with “bounds for error”



GUI Before Implementation of Fortune’s Algorithm



First Implementation of Fortune’s Algorithm with Maps



Final Implementation of GUI

# **Testing Plan & Results**

* 1. Testing for VIR-Atlas was conducted throughout development at each major milestone.
     1. Functional Tests were used as the GUI was coming to completion to check that system responses to erroneous conditions fell within acceptable parameters. Activities such as loading maps with no legends, exiting out of menu options without completing them, changing maps after adding annotations, all represent different functionalities which should be idempotent operations that were tested for.
     2. During feature building, team members that collaborated updated one another on bugs found during test runs and dealt with or noted them in our project server on Discord.
     3. Code Inspection assisted in the finding of a parsing bug concerning the STELLA data.
     4. Usability Testing leading up to the project demo highlighted bugs with the annotation, save/load/map-display/and scrolling features that were then addressed.
     5. After our team moves the project to Github to be available as open-source we will start implementing Github’s Issues feature for more efficient bug logging and the Continuous Integration/Deployment features for more robust testing.

# **Project Status and Summary**

## **Project Status**

* + 1. Annotations are still a little buggy due to many problems we had encountered with Tkinter. Right now annotation object names appear as the annotations saved in the main “Annotations'' window . We also had a lot of problems with Tkinter not being consistent with displaying its main window in the GUI for a Linux OS vs a Windows OS. For now, we’ve left these problems as they are, and in the future we would like to switch the GUI to something not Tkinter based. Annotations also do not dynamically scroll with the map.
    2. Our interim report demo with Paul Mirel, led to our team speaking to other researchers utilizing the STELLA hardware, and allowed for us to decide to change our plans to implement 8 maps instead of 3 originally, based on how a researcher would be able to utilize this type of data. All 8 of our maps display correctly in VIR-Atlas and dynamic scrolling is still available. This is also when we officially decided to implement annotations and satellite images, which were originally going to be characteristics of VIR-Atlas if we had time to implement them.
    3. Otherwise, VIR-Atlas visually displays all of the data we feel will allow researchers and educators to really utilize this software.

## **Difficulties Encountered**

## Tkinter

Tkinter, the standard GUI library for python, ended up having a lot of limitations to the style and implementation that we wanted for our graphic user interface. We did the best we could with what we had, but it significantly slowed down the progress with the front end. In the end we were able to complete most of what we wanted (minus the bugs in annotations) but this is definitely not a GUI library any of us would like to use again.

## Satellite Images

We first looked at the Google Maps API for our satellite imagery. However, this was behind a steep paywall which prevented us from using this as a solution. We instead used the sat-search Python package that uses satellite images from the Sentinel2 satellite. The resolution (10m-60m vs. Google Map’s 15cm) of this satellite was significantly lower than desired, but we were unable to find another viable solution.

## Unfamiliarity with Project Coding Language

VIR-Atlas was completely written in Python because we wanted a language that was flexible, easy to pick up, and powerful with respect to array handling. Python fit those requirements, however it was still a difficult task to learn a new language while developing in it and juggling classes at the same time. Thankfully there are an immense amount of resources available for learning how to code with Python that made the process possible.

## Color Theory (Spectral to RGB)

While a standard for color generation was used, these standards differ depending on the white point used, the device recording the data, and piecewise gaussian distributions. Color went through many repetitions of debugging to find that the program needed to be calibrated to STELLA’s range of light recorded and the white point initially used did not match the translation matrix. The constants for each standard did not necessarily match across resources and were never all in the same resource.

## Map Generation Algorithm

VIR-Atlas’s map generation algorithm went through 3 different iterations, as no members of the team were knowledgeable in graphics. Initially Triangles were used that calculated the color of their neighbor to get their color, but this couldn’t consider the altitude at which the data was taken, nor did it blend colors organically. Circles considered altitude, but overlapped, so some data was no longer visible. Finally, Fortune’s algorithm was selected, which develops equal euclidean distance polygons based on seed points.

## Takeoff and Landing Data

One of the largest issues in pairing data was the fact that STELLA and the Drone take data at different rates. The Drone pings roughly 10.6 times faster. Since the devices were not started at the same time either (STELLA had to be activated first) this created a time mismatch between the devices as well. The solution was to pair the first data point based on takeoff times and pair using the delta time since the first point. Takeoff needed to be chopped for a number of reasons, one being bad data from STELLA (All color readings appeared black due to shadow) and altitude differences. Some constants were developed for handling removing the bad data from takeoff.

## **Journal of Project Activities**

Meeting minutes are recorded every meeting on the assignment of tasks, the discussion of implementation, and debugging.

[Meeting Minutes 02/01/2021 - 04/23/2021](https://docs.google.com/document/d/19nEsHZ1n2jDXi1nQFxlGtikKotAZDGLy_QSUMqyJjls/edit?usp=sharing)

## **Individual Contributions**

* + 1. **Marisa:**

Project Lead: delegated tasks each week and kept track of goals and deadlines throughout the semester. Facilitated communication about project progress in meetings and discord chat.

Frontend: Majority of Annotation Development, Functional Testing

Backend: Data Management for GPS and STELLA data, Map Generation

* + 1. **Tyler:**

Frontend: Displaying all Components in GUI, Default Display, Loading Files, Overall Architecture, User Interface Design, Frontend Class Designs, STELLAFrame, Menubar Design, Annotation Design, Functional Testing, General PEP8 formatting and file cleanup

* + 1. **Timothy:**

Frontend: Map Legend Development, Functional Testing

Documentation: Gitlab User Guide

* + 1. **Franklin:**

Frontend: Scrolling/panning/zooming functionality for STELLA and satellite windows, satellite frame implementation, GUI design

Backend: Satellite image retrieval

* + 1. **Sophia:**

Frontend: General debugging, Temperature and SVA Legend, Connection between Front and Back, StellaFrame swapping

Backend: Spectral data to RGB Color, False Color, Vegetation Index Color, Developed Data point pairing algorithm, Implemented Fortune’s algorithm for Map generation

Data Acquisition: Built STELLA unit, Flew Drone (Took Data)

* + 1. **Tenise:**

Frontend: Window Design, Annotation Design, Annotation Development, Menubar Design, and Menubar Development

Project Organization: Wrote all Meeting Minutes, keeping track of tasks and goals delegated each week. This helped facilitate communication between meetings.

## **Project Schedule**

## Chart

# **Appendices**

* 1. Resources:
     1. Vegetative Indexes and their calculations:

<https://www.usgs.gov/core-science-systems/nli/landsat/landsat-surface-reflectance-derived-spectral-indices?qt-science_support_page_related_con=0#qt-science_support_page_related_con>

<https://earthobservatory.nasa.gov/features/MeasuringVegetation/measuring_vegetation_2.php>

* + 1. Voronoi Polygons and Fortune’s Algorithm:

<https://www.cs.umd.edu/class/spring2020/cmsc754/Lects/lect11-vor.pdf>

<http://www-cs-students.stanford.edu/~amitp/game-programming/polygon-map-generation/>

* + 1. Color Calculation:

<http://www.brucelindbloom.com/index.html?ColorCalculator.html>

<http://jcgt.org/published/0002/02/01/paper.pdf> <https://www.oceanopticsbook.info/view/photometry-and-visibility/from-xyz-to-rgb>

* 1. Source code: <https://git.cs.nmt.edu/fkeith/vir-atlas.git>