INDY – 5 DEVELOPING A DATA VISUALIZATION APP FOR CAVE TEMPERATURES AND MICROCLIMATES

CS 4850 – Section 01 – Spring 2025

Caroline Roberson, Nathan Karg, Hayden Harper, Abdalla Ugas

Dr. Sharon Perry

April 20, 2025

GitHub Repo: https://github.com/CaveVis/GraphBat-Data-Visualizer

Project Website: https://cavevis.github.io/GraphBat-Data-Visualizer/project_info.html

Stats and Status	
LoC	~4000
Components/Tools	Python 3.12, PySide, PyQt, Numba, Numpy, matplotlib
Hours Estimate	360
Hours Actual	460

Table of Contents

Project Overview

Summary

Speleology, the study of caves and karst features, often includes placing sensors in a cave and recording data over time. This could include humidity, air temperature, water temperature, wind speeds, and more. Scientists need to be able to take this data and convert it into digestible graphics for various audiences. GraphBat is a desktop data visualization application designed for speleology and similar fields that bundles common graph types with a unique heatmap tool which few comparable apps provide. It is written in Python and is intended as an open-source tool available for use and extension by the scientific community. The heatmaps offer two data interpolation methods—inverse distance weighting and linear interpolation—to visualize the spread of data across a space using a real-world map and sensor data relative to the space. GraphBat aims to expedite scientific analysis and facilitate the presentation of results across many fields of subterranean study.

Objectives

- **Ensured complete delivery** of the product by developing and adhering to complete project documentation
- **Enhanced individual productivity** of speleology research projects in professional and academic settings by delivering a robust, efficient, and well-tested product
- **Maintained product quality** by performing frequent, rigorous testing of all project components
- **Validated product requirements** by conducting user testing with individuals from related fields of work
- **Ensured product efficiency** by collecting original data and testing the product as an end user

Project Links

GitHub Repo

The source code and user manual can be accessed from this link:

https://github.com/CaveVis/GraphBat-Data-Visualizer

Project Website

Project info, deliverables, source code, and other information can be viewed here:

https://cavevis.github.io/GraphBat-Data-Visualizer/project_info.html

Deliverables

At the end of development, we delivered the following items:

- **A final report package** that includes a document describing the development and design process from start to finish; the complete source code for the application; a video presentation that demonstrates common usage of the application; and a website that showcases the previous components
- **A working data visualization application** that provides all of the functions outlined in the project requirements and that runs with minimal error on most desktop machines
- **A video demonstration** showing the major features of the application and how to use them along with discussing the development process

Definitions & Acronyms

Term	Definition
Bin	A bar on a histogram graph that represents a range of values. The taller the bar, the more frequently the range occurs in a dataset.
Heatmap	A representation of data in the form of a map or diagram in which data values are represented as colors
Histogram	A diagram consisting of rectangles whose area is proportional to the frequency of a variable and whose width is equal to the class interval.
IQR	Interquartile Range, the middle 50% spread of the data. Defined as the difference between the 75 and 25 percentiles of the data
Speleology	The study of caves.

Requirements

Functional Requirements

1. **Projects**

1.1 The system shall allow the user to assign data to a new project.

1.1.1 The system shall allow the user to import zero or more datasets in . .CSV format into the project.

1.1.2 The system shall allow the user to import zero or more cave maps in .JPG or .PNG format.

1.2 The system shall allow the user to assign a new project a name.

1.2.1 The system shall allow the user to input a name that is at most 40 characters.

1.2.2 The system shall allow the user to input a name that may contain alphanumeric characters, hyphens, underscores, and parentheses.

1.2.3 The system shall notify the user if the length of their project

name exceeds 40 characters with the following notice: "Project name cannot exceed 40 characters."

1.2.4 The system shall notify the user if an invalid character is input as a name with the following notice: "<character> is not allowed."

1.2.5 The system shall notify the user if another project of the same name exists with the following notice: "Another project exists with this name. Please enter a unique name."

1.3 The system shall allow the user to assign a new project a project description.

1.3.1 The system shall allow the user input a project description that is at most 500 characters.

1.3.2 The system shall notify the user if the length of their project description exceeds 500 characters.

1.4 The system shall allow the user to save their new project.

1.4.1 The system shall save the new project only if the project name and description are valid.

1.4.2 The system shall save the new project whether or not there is a dataset attached.

1.4.3 The system shall notify the user if the project name or description are invalid when they attempt to save a new project.

1.4.4 The system shall notify the user if the project name is less than 1 character with the following notice: "Project name is blank."

1.4.5 The system shall notify the user if the project does not have a dataset attached at the time of saving with the following notice: "Project does not contain any data (data can be added later). Save new project?"

1.4.6 The system shall commit the new project when the user selects the save option.

1.4.7 The system shall require the user to confirm their decision to save if the project does not have a dataset attached.

1.4.8 The system shall require the user to confirm their decision to save if the project has a dataset attached.

1.5 The system shall allow the user to modify an existing project.

1.5.1 The system shall allow the user to modify an existing project's name, description, and attached data.

1.5.2 The system shall require all changes to an existing project to adhere to the same validation standards as during project creation.

1.6 The system shall allow the user to delete an existing project.

1.6.1 The system shall require the user to confirm their decision to delete an existing project twice before committing.

1.6.2 The system shall remove all data related to a deleted project once the user has confirmed their decision to delete.

1.7 The system shall allow the user to load an existing project.

1.7.1 The system shall display a list of existing projects in order of creation.

1.7.2 The system shall load all data associated with the selected project.

1.7.3 The system shall allow the user to load a different project at any time.

1.7.4 The system shall display the name of the currently selected project across all screens.

1.7.5 The system shall display the number of attached data and image files per project.

2. Importing Data

2.1 The system shall allow the user to import a . .CSV file as one of a project's datasets.

2.1.1 The system shall import . .CSV data with a default format of Date (MM/DD/YY), Time (HH:MM), and Temperature if there is no prior user-defined format.

2.1.2 The system shall read the first row of a . .CSV file and parse the labels.

2.1.3 The system shall divide all columns of data in a . .CSV and store them.

2.1.4 The system shall allow the user to customize the labels of read data.

2.1.5 The system shall require the user to enter a label using solely alphanumeric characters, hyphens, underscores, and parentheses.

3. Visualizations

3.1 The system shall allow the user to save created visualizations to the application database.

3.2 The system shall allow the user to load any of a project's pre-existing visualizations.

3.3 The system shall allow the user to delete any of a project's pre-existing visualizations.

3.3.1 The system shall require the user to confirm their decision to delete a visualization twice before committing the change.

3.4 The system shall allow the user to visualize a project's dataset as a line graph.

3.4.1 The system shall draw the graph only if for each axis there exists a valid column type that has been selected by the user

during column selection.

3.4.2 The system shall require the user to select one of the following valid column types, if at least one exists in the dataset, to use as the x axis: Time or Date.

3.4.3 The system shall require the user to select one of the following valid column types, if at least one exists in the dataset, to use as the y axis: Temperature or Humidity.

3.4.4 The system shall draw the graph only if for each axis there exists a valid column type and it is selected by the user.

3.4.5 The system shall add every nth point to the graph.

3.4.6 The system shall draw a line from left to right between each point consecutively.

3.4.7 The system shall label the axes with the user-selected labels from the dataset.

3.4.8 The system shall display the x and y axis intersection value for each point when it is clicked on.

3.4.9 The system shall update and redraw the graph whenever the user modifies an element.

3.4.10 The system shall determine the range of values for each column assigned to an axis.

3.4.11 The system shall denote each axis with 9 tick marks, each marking 10% of the data's range, with the opposite axis marking the bottom 0%.

3.4.12 The system shall round all tick marks to the nearest 0.5.

3.5 The system shall allow the user to visualize a project's dataset as a box plot.

3.5.1 The system shall draw the graph only if for each axis there exists a valid column type that has been selected by the user during column selection.

3.5.2 The system shall label the axes with the user-selected labels from the dataset.

3.5.3 The system shall update and redraw the box plot whenever the user modifies an element.

3.5.4 The system shall prompt the user to attach a dataset to the project if none exists.

3.5.5 The system shall calculate and display the interquartile range (IQR) for the box plot.

3.6 The system shall allow the user to visualize a project's dataset as a histogram.

3.6.1 The system shall draw the graph only if for each axis there exists a valid column type that has been selected by the user during column selection.

3.6.2 The system shall label the y axis as Frequency.

3.6.3 The system shall draw the graph only if for the x axis there exists a valid column type and it is selected by the user.

3.6.4 The system shall update and redraw the graph whenever the user modifies an element.

3.6.5 The system shall prompt the user to attach a dataset to the project if none exists.

3.6.6 The system shall prompt the user to attach a dataset with a valid column type if the current dataset does not contain at least one valid column type for the x axis.

3.6.7 The system shall divide the data into bins and plot them on the x axis.

3.6.8 The system shall calculate the frequencies of each bin and plot them on the y axis.

3.7 The system shall allow the user to visualize a project's dataset against the project's cave map.

3.7.1 The system shall prompt the user to attach a dataset to the project if none exists.

3.7.2 The system shall prompt the user to attach a cave map to the project if none exists.

3.7.3 The system shall generate a grid of tiles.

3.7.4 The system shall allow the user to define the number of tiles on the grid.

3.7.5 The system shall display the project's cave map and allow the user to draw a selection area to use in the visualization.

3.7.6 The system shall allow the user to draw a selection area to exclude from their current selection.

3.7.7 The system shall allow the user to draw a selection to include in their current selection.

3.7.8 The system shall hide all grid tiles that lie entirely outside of the selection area.

3.7.9 The system shall require the user to select a tile to represent the collection location of each dataset.

3.7.10 The system shall mark the selected tiles as sensor tiles.

3.7.11 The system shall calculate the value for each tile based on its distance from each sensor tile, the alpha value, and value from the dataset at a timestamp

3.7.12 The system shall calculate distances from each sensor to each point linearly

3.7.13 The system shall calculate distances from each sensor to each valid point using Dijkstra's Algorithm

3.7.14 The system shall display the value of a tile when it is hovered over.

3.7.15 The system shall use the default colors of purple and pink to denote the low and high ends of the data range respectively.

3.7.16 The system shall color each tile based on its value.

4. Exporting Data

4.1 The system shall allow the user to export visualizations as images.

4.2 The system shall allow the user to export visualizations as PDFs.

4.3 The system shall allow the user to export heat map visualizations as videos.

5. Application Settings

5.1 The system shall allow the user to customize the application's settings.

5.1.1 The system shall allow the user to choose between a light and dark theme.

5.1.2 The system shall draw all UI elements according to the currently selected theme.

5.1.3 The system shall allow the user to choose from a list of fonts to display the application's text in.

5.1.4 The system shall allow the user to choose from a list of fonts to display graph labels and titles in.

5.1.5 The system shall draw all UI text elements according to the currently selected fonts.

5.1.6 The system shall allow the user to switch between windowed and fullscreen mode.

Non-Functional Requirements

1. Usability

1.1 The system shall consistently render the position, order, and appearance of critical UI elements, such as save/cancel buttons, across all windows.

1.2 The system shall use a combination of color and text/symbols to indicate

the purpose of critical UI elements, such as save/cancel buttons, across all windows.

1.3 The system shall be usable by the average person with one hour of informal training or less.

1.4 The system shall allow the user to navigate to anywhere in the application in 3 pages or less.

2. Performance

2.1 The system shall render all visualizations in at most 30 seconds.

2.2 The system shall import all data in at most 5 seconds.

2.3 The system shall export all data in at most 30 seconds.

Design

Assumptions

Data Sensitivity: It is expected that users have enabled simple desktop security (e.g. requiring a password to sign onto the machine, appropriate time-out measures) and are not inputting high-risk or confidential information into the program.

- **Data is stored locally** on the machine and will require no internet functionality.
- **Data is not hidden or encrypted** and the program requires no log in.

Design Constraints

Environment

The expected operating environments for this program are modern desktop machines running recent versions of Windows operating systems.

- The average machine is expected to have 8 GB of RAM and have at least 500 MB of storage available to contain the program executable and data outputs.
- Supported operating systems were Windows 10 or later. The program may run on other operating systems or earlier versions of Windows, but it will not be specifically tested and guaranteed.

User Characteristics

The intended users for this program are speleologists, but it is possible that users from fields such as mycology or other, unrelated fields may utilize the program for similar projects. Users may or may not be experts in their field and may have varying levels of computer literacy. The program's interface must be designed with these users in mind.

- The program must permit users with **greater expertise and skill** to use it to its full effectiveness and provide training or tutorials for inexperienced users.
- The average user is expected to have **basic computer literacy** (i.e. how to locate, open, delete, and save files; how to run and operate an executable program; and how to interact with a machine's GUI).
- The average user is not expected to have **advanced computer skills** or any programming ability (e.g. how to build or run a program from the command line).

Performance Requirements

The computations for the various visualizations are relatively low, even for datasets with tens of thousands of logs per sensor. The graphing of sensor data was not an issue from a performance perspective, as all plotting was performed in less than five seconds. The main consideration for performance requirements is the data processing speed and efficiency for the heat map generation routines. At earlier points in the project, generating a single heat map would take over ten seconds for an average resolution and would require about 20 GB of storage for 20,000 heatmaps.

Architectural Strategies

Platform

The application was initially set to be developed in **Unity version 2023.2.20f1**. Although Unity is better known for its use in game development, it offers robust and user-friendly UI development tools that would have significantly accelerated the application's rate of development and reduced the amount of time spent by team members on learning a more complex framework. For these reasons, it is a popular choice in data science for creating interactive visualizations and simulations.

The decision to pivot to developing the application with **Python 3.13.0** and its associated tools was driven by a number of factors, namely the need for more flexible data manipulation and graphing than what Unity readily offered, easier integration with scientific libraries, and the ability to leverage the existing Python experience several members of the development team already had. Additionally, Python is the more commonly used programming language used in the scientific community, who may have little experience with computer science, Unity, or other, more complex languages.

Software Extension

Each feature–for example, separate graphs–was designed to operate relatively independent of one another. Once the basic components such as importing, exporting, and saving data were added, extending the application to include additional types of graphs or other features was straightforward. Components were designed with low coupling and high cohesion in mind to facilitate future extension by either the original development team or members of the open-source community. This approach also sped up the software validation and verification process.

Error Detection & Recovery

To ensure that the application is flexible and resilient when handling errors, the following processes were implemented.

- **File Input Validation** - verify the existence of input files before attempting to read them, ensure file permissions allow read access, and check for valid file extensions
- **Data Format Validation and Adaptation** - support multiple input file formats and normalize data to a standard schema for display purposes
- **Missing or Corrupted Data Handling** - eliminate missing values from the data to minimize discrepancies in the calculations
- **User Feedback and Debugging Tools** - provide the user with a summary of data issues encountered and potential recovery steps applicable

System Architecture

The system is responsible for the following functions:

- **Creating, deleting, and modifying** projects and project data
- **Importing data** including . .CSV datasets and .JPG, .XPG, and .PNG datasets
- **Generating visualizations** such as graphs or other high-fidelity visualizations
- **Exporting data** including .XPG, .JPG, or .PNG images; . PDF reports; and .MP4 video files

The following classes are responsible for storing application data:

- The **ProjectManager** class stores information about projects, such as a project's name, description, project ID in a .json file upon project creation, and handles creation of various project directory folders.
- The **DataProcessor** class stores data pertaining to the cave a project is created for, such as .JPG or .PNG cave maps and . .CSV dataset, by copying their files to the appropriate project folder.

The application's functionalities are handled by these components:

- The **data_processor** component handles importing external data provided by the user, validating the data, and creating an instance of the DataProcessor class to store it
- The **Matplotlibcanvas** component handles saving images of graphs created by the application and exporting them to the user's images folder
- The **display_canvas_in_frame** component handles plotting processed data into graphs or other visualizations
- The **ProjectManager** component handles creating, modifying, deleting, and reading projects
- The **Mainwindow** component handles displaying pages, navigating pages, and updating UI elements

System Overview

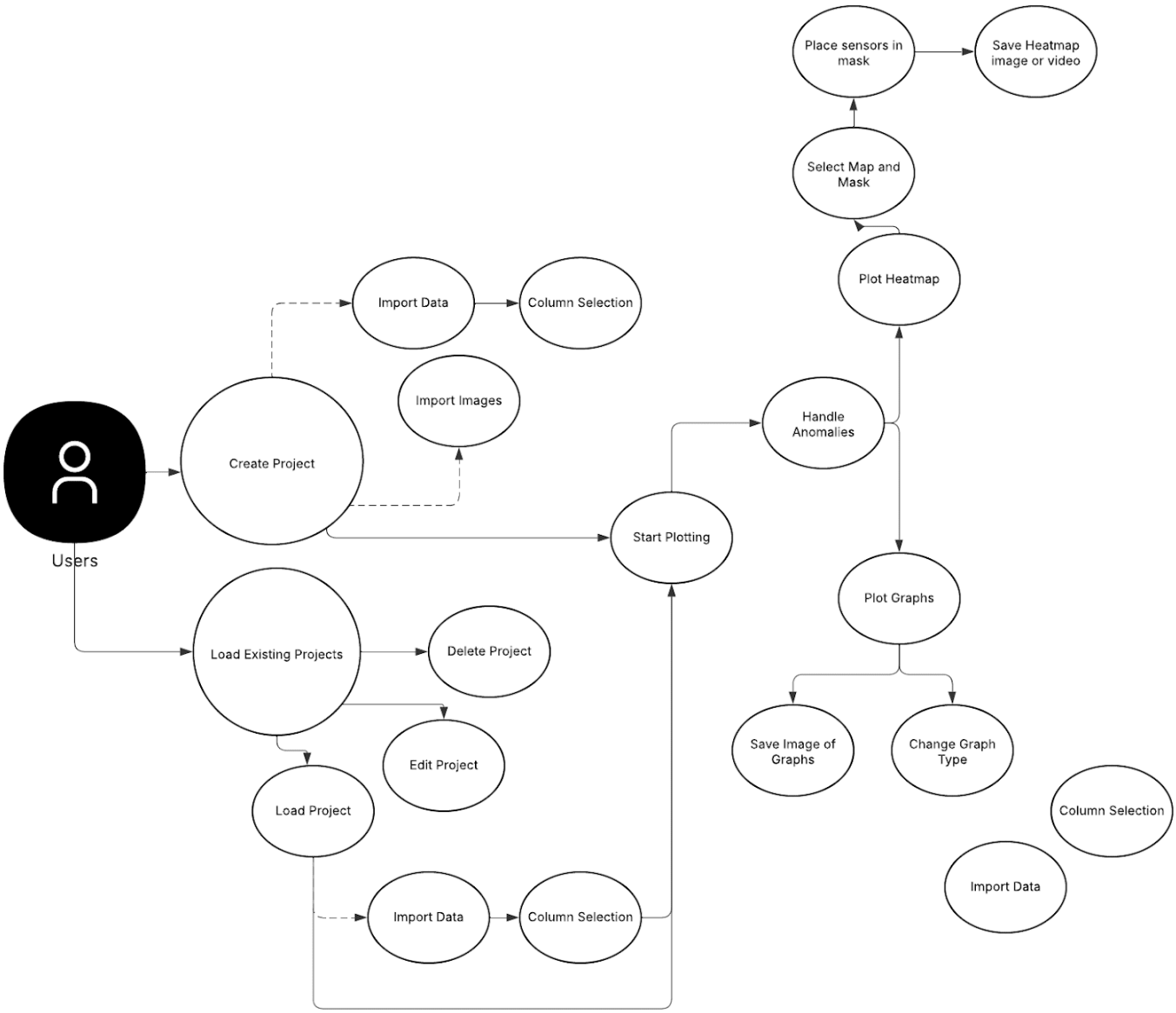

Figure 1: A use case diagram demonstrating GraphBat features

The application opens to the main menu where the user has the option to create a new project or load an existing one. For project creation, users are required to give the project a valid name, but may optionally add a project description and import data. If they are importing data upon project creation, then they will be prompted to choose which columns of the dataset to use as the index and data columns, respectively.

If users choose to load a project, they will choose from a list of valid projects that are on their computer, if any. A listed project will include the project title, number of attached data and image files, and the date of last accessing the project. At this point, users also have the option to edit information or delete a project from their list. Upon loading a project from the list, the system will attempt to load the project information from the .json file initialized during project creation, as well as load the attached data files. If project loading is successful, then the user will be taken to the project homepage.

The project homepage offers a variety of different visualizations that the user may choose to view their data in. The graphs included for basic visualizations currently include line and bar graphs, histograms, and box plots. The user also has the option to view their data as a heatmap overlaid on the cave map but are required to first import a valid map, draw a mask outline for visualizing, and place their sensor(s) in their respective locations on the map. After visualizing their data, users have the option to export their data either as an image of the graphed data or heatmap as well as a video of the entire time range of the heatmap, which will be saved in the project's images and videos folders, respectively.

Detailed System Design

Project Manager Class

Definition: A ProjectManager is the highest-level data in the system. It contains attributes storing its name, description, time of creation, and last time modified.

Responsibilities: Creating, modifying, and deleting project files.

Data Processor Module

Definition: The data_processor.py module is designed to handle the processing of .CSV and file images with a focus on data column selection, anomaly detection, and preprocessing.

DataProcessor - ColumnSelectionDialog Class

Definition: Creates a dialog that allows users to select columns from .CSV files. It extends QDialog from the PySide6 library.

Responsibilities: Allowing users to select index and data columns from .CSV files, providing functionality to rename selected data columns, and ensure the user selects at least one index column and data column prior to accepting dialog.

DataProcessor - AnomalyDialog Class

Definition: Creates dialog for displaying and handling anomalies detected in dataset. It extends QDialog from the PySide6 library.

Responsibilities: Displaying detailed information about detected anomalies, providing options for users on how to handle detected anomalies (remove, ignore, or view), and displaying a table of anomalies with timestamps and values.

DataProcessor - DataProcessor Class

Definition: Handles the core functionality of the data processing system, including file selection, data reading, and anomaly detection.

Responsibilities: Handles the selection and reading of .CSV and image files, reads and preprocesses data from .CSV files, including resampling and anomaly detection, merges preprocessed data from multiple files into a single DataFrame, and maintains the state of sensors and their data.

Heatmap_Widget Class

Definition: Heatmap_Widget is a custom QWidget that displays an interactive, timestamp based, interpolated heat map overlaid on an image using Matplotlib.

Responsibilities: Display an image and a heatmap while allowing the user to scroll through the selected timestamp range, save specific heatmaps, and save a video of the entire time range.

Draw_Tool Class

Definition: Draw_Tool is a custom QWidget that allows a user to upload an image and draw a mask over it using a variable brush size with eraser toggle.
Responsibilities: Provide the user a way to upload an image to use as a reference for drawing a mask. Provide the user with a brush tool to designate the valid portions of the map. Allow the user to change the size of the brush tool and toggle it to be an eraser. Export the mask to be used in the heat map visualization.

Main Module

Definition: The main.py module is designed to handle UI initialization and management, connection of PySide elements, and application settings.

Main - MainWindow Class

Definition: Primary application window that inherits from QMainWindow; it initializes and manages the entire UI, handles user interactions, and coordinates between different elements

Responsibilities: Initializes and configures the UI components, manages application settings (theme, full screen mode, font preferences). Also works to display processed datasets with the different aforementioned visualization types

Main - MatplotlibCanvas Class

Definition: A custom canvas class that inherits from *FigureCanvasQTAgg* to embed Matplotlib plots within the Qt application.

Responsibilities: Initialize a Matplotlib figure with appropriate dimensions and DPI within the given frame, provide a drawing surface for various plots and visualizations, and enable integration between Matplotlib and Qt widgets.

Main - CustomNavigationToolbar

Definition: Extends Matplotlib's default *NavigationToolbar2QT* to add custom functionality for time range selection

Responsibilities: Provide standard Matplotlib navigation controls (zoom, pan, etc.), add a custom time range adjustment button, implement the time range selection dialog, and finally apply selected time ranges to the current visualizations.

Development

Methodology

The development methodology chosen for this project is an **informal iterative/incremental** approach with some elements from the **kanban methodology**. The program contains a variety of modular components that function relatively independent of each other. These components can be developed one at a time and added to the program as resources permit. They can be adjusted or removed based on user feedback without disrupting the program. An iterative/incremental approach allows individual functions to be conceptualized, developed, implemented, tested, and improved at separate stages. This ensures only finished, verified functions are added, and anything that cannot be completed by the deadline is omitted. Adding features sequentially facilitates the process of determining whether there is enough time and reason to add a new feature. Taking an iterative approach to testing also accelerates

identifying and solving bugs that crop up when a new component is added. Each iteration, new tests should be designed for the new component, and all existing tests should be re-run. During each increment, the major tasks should be divided into subtasks and tracked on a kanban board.

Other development methodologies were considered and ultimately dismissed. One option was the **waterfall method**. This method, while on average effective for most projects, is a poor fit due to its rigidity. The program must be developed in a short timeframe and will need a few rounds of user testing. The waterfall method will force all of the testing and feedback to be conducted at the end of the development period, at which time it would be too late to make critical changes. **Scrum** was another option but was ultimately dismissed because proper scrum methodology requires at least one team member to be familiar with the technique and to act as scrum master, which was not a possibility; additionally, its requirement of daily standups was unrealistic due to the team's conflicting commitments.

Conceptualization

Visualizations

Graph Library Selection

To visualize data, the project needed a library capable of drawing graphs that plot a large number of data points quickly, even on machines with average computing power. Additionally, the library must be capable of interfacing with **PySide** and/or **PyQt** libraries, preferably as easily as possible. As none of the developers on this project have significant experience with any single plotting library, finding an option with high availability of tutorials and a relatively shallow learning curve was the priority to spend time effectively.

Option 1: Matplotlib

Pros: Matplotlib is a common Python graphing library that interfaces well with other common Python libraries, including Pandas. Matplotlib graphs are supported by **PySide**, making inserting plots generated by the library into the application UI relatively simple. It is also highly customizable, being one of the more versatile graphing libraries available, but its default graph style is already aesthetically-pleasing. Given its popularity, there is a large amount of documentation available. Matplotlib code is also relatively simple, which reduces the time investment required to learn it.

Cons: Matplotlib has some issues with larger datasets and may lag when rendering a high volume of data points, particularly on lower end machines. Although universal good performance is important given the wide variety of machines that may be used by different users, the average dataset is expected to be reasonably small. The resulting drop in performance should therefore not be significant.

Option 2: PyQtGraph

Pros: PyQtGraph is built for use in scientific applications and is designed with **PySide** and **PyQt**, making inserting the renders into the application UI simple. Additionally, PyQtGraph uses the GPU for intensive computing, which cuts down on rendering time. This could make it a prime option if the average dataset is expected to be large.

Cons: PyQtGraph is younger than Matplotlib and not used quite as commonly, meaning there were fewer tutorials and documentation available. Learning to use it effectively may therefore take extra time. Additionally, PyQtGraph's default graph style is not the most pleasing and will require significant editing, but regardless this aligns with the project's goal of having a unique, well-designed UI.

Option 3: QWidget, QPainter

Pros: The QWidget class, a part of the **PyQt** library, offers the **QPainter** widget which allows for drawing individual lines and other shapes. This would provide the highest degree of control over exactly how the graph renders and looks. The lack of fancier rendering may also reduce the computational power required for even small graphs.

Cons: QPainter's lack of substantial existing functions for rendering graphs means a significant amount of time would need be dedicated to developing the tool from scratch.

Conclusion

Matplotlib was selected as the primary option for its simplicity, versatility, and the availability of documentation and other resources. Additionally, since Matplotlib interfaces easily with **Pandas**, it was the better choice for ensuring passing input data to the graph engine is as smooth as possible. PyQtGraph was selected as the backup option in the event Matplotlib is insufficient for the project. Although Matplotlib may not handle large datasets as well, this weakness can be offset by efficiently chunking data.

Graph Types

The user will need to be able to see the visualizations of the sensor data in a variety of ways. Some core statistics that a user may want to visualize could include the mean, low, and high temperatures for a sensor over a period of time. The average rate of change and standard deviation could also be a useful statistic to present. In addition to viewing information about the data collected from a single sensor, it would also be helpful for cave research purposes to be able to compare aggregated statistics for regions of the cave.

In addition to those key statistics, users would also be able to see different graph types for the visualizer sensor data. Users would be able to compare aggregated data such as the previously mentioned stats with bar charts as well as box plots/histograms to compare the distributions of data readings across different sensors. When tracking sensor measurements over time, it is additionally useful to be able to use time-series

line charts, such as line graphs. Multiple sensors could be compared with each sensor having a different line with a matching legend for accessibility purposes.

Although the project focuses on temperature data, a potential goal is to allow users to import multiple types of data, such as humidity. To view the potential correlation between those two parameters, a scatterplot could highlight potential relationships. Bar graphs could also be used for viewing aggregates of multiple types of data.

Graph Implementation

In order to integrate Matplotlib into the Pyside6 / PyQt5 application, the data stored in Pandas DataFrames would need to be accessed. Pandas integrates natively with Matplotlib, allowing relative ease with passing entire columns of data to Matplotlib plotting functions. Actual usage of Matplotlib would begin with embedding its FigureCanvasQTAgg within a Qt widget. The UI layout can be designed in QTCreator for the controls like buttons or dropdowns. With a custom widget class, the Matplotlib figure can be initialized and attached to the canvas.

After computing the previously mentioned statistics for indicated sensors, Pandas would pass this data to the plotting functions that Matplotlib provides, such as plt.plot() or plt.scatter(). Leveraging PySide signals such as button clicks or combo-box selections would allow dynamic updating of plots by clearing axes and regenerating visuals with filtered data.

Interpolation

A major part of the application is the heat map visualizer, which allows the user to input a map of a given cave or area along with sensor data and view an interpolated heat map showing the distribution of the data values throughout the area. The map is divided into a grid and the sensors are assigned locations that approximately mirror their real-life locations in the cave. The user will use a draw tool to indicate where the interpolation should occur. Then, using the data from each sensor, the value for each grid tile is calculated using some interpolation method, chosen and customized by the user.

The different interpolation methods like inverse distance weighting, inverse distance weighting with area weighting, and linear interpolation, will all provide different visuals to accommodate more types of data.

Weighting Values with IDW

The inverse distance weighting (IDW) formula is as follows

$$\frac{\left(\frac{1}{(d_1)^\alpha}\cdot z_1\right)+\left(\frac{1}{(d_2)^\alpha}\cdot z_2\right)+\left(\frac{1}{(d_3)^\alpha}\cdot z_3\right)\ldots+\left(\frac{1}{(d_n)^\alpha}\cdot z_n\right)}{\left(\frac{1}{(d_1)^\alpha}+\frac{1}{(d_2)^\alpha}+\frac{1}{(d_3)^\alpha}\ldots+\frac{1}{(d_n)^\alpha}\right)}$$

Figure 2: Inverse distance weighting formula

where n is the number of sensors, d_n is the distance from a point to sensor n, z_n is the value of sensor n, and α is the power parameter. Note that α is used to adjust the formula to prefer higher or lower values, which corresponds to further or shorter distances. The cave map was laid out on the x, y axes used to calculate distance. The weights for each sensor $w_n = \frac{1}{(d_n)^\alpha}$. The distance between each sensor and a given point can be calculated using a pathfinding algorithm.

The value for each sensor $z_n = \frac{\sum_{n=1}^{i} w_i z_i}{\sum_{n=1}^{i} w_i}$

Linear Distance

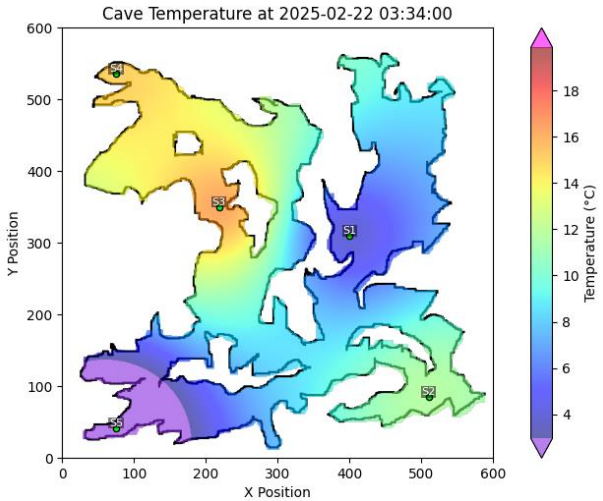

Figure 3: Results of heatmap using linear interpolation

Dijkstra's Algorithm

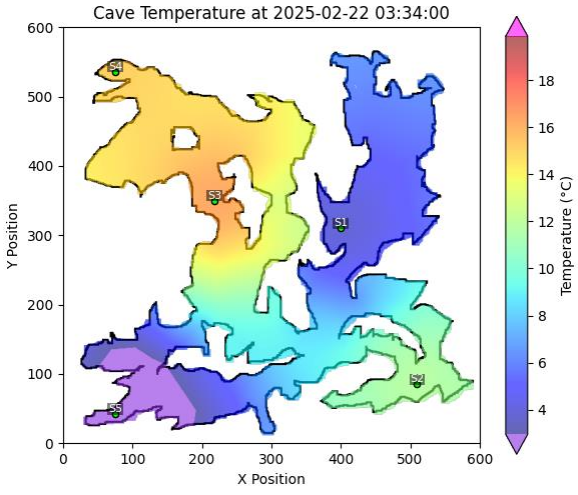

Figure 4: Results of heatmap using Dijkstra's algorithm

User Interface

The application UI was created with **PySide6** and **PyQT5**. These libraries are robust and designed to streamline the process of creating UI. Tools like Qt Creator and Qt Designer provided a drag-and-drop editor that further sped up UI development, allowing more time to iterate over versions of the application UI. The user interface was designed iteratively. Each screen was first prototyped on paper to clarify the general layout and identify the most important components of the screen.

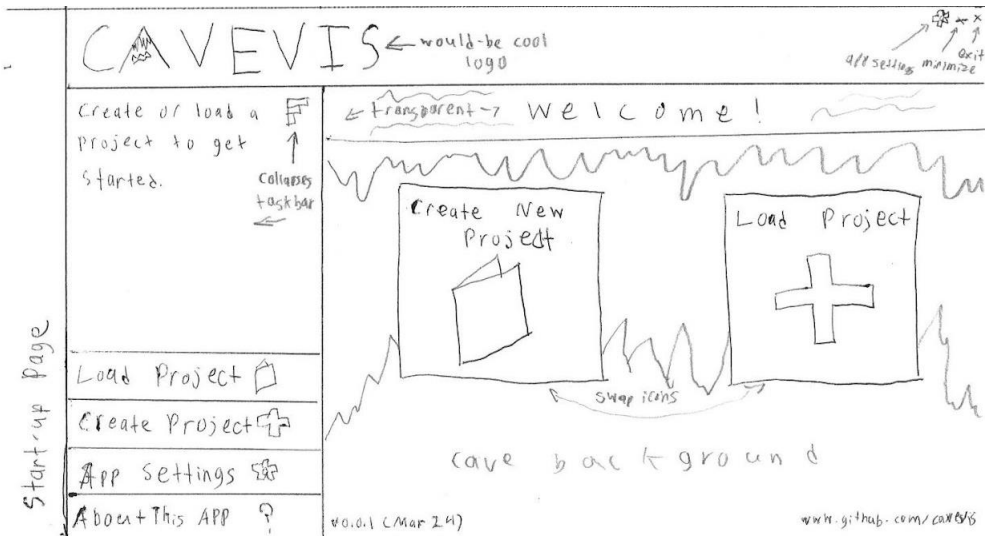

Figure 5: Paper prototype of GraphBat's start screen

Then, the screen was prototyped in Marvel to see how the design applied to it. The original color scheme was primarily blue and black, then later changed to brown and black. The brown iteration would be further developed to refine the shades used.

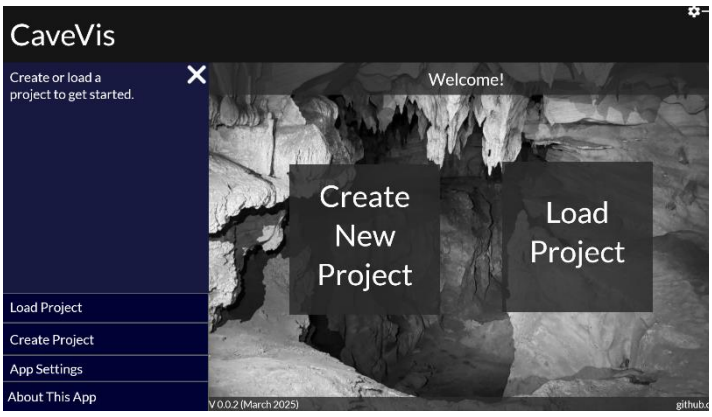

Figure 6: Marvel mockup of GraphBat using a blue color scheme

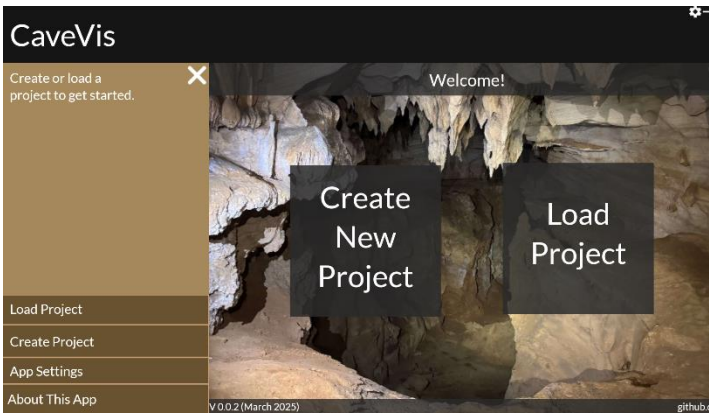

Figure 7: Marvel mockup of GraphBat using a brown color scheme

The final color scheme and fonts were chosen using color selection from the colored cave photo in the background of the prototypes shown above. These were used in the prototype application developed using Python's Qt library. In addition, the original cave photo was redesigned to be more abstract and less distracting to the user.

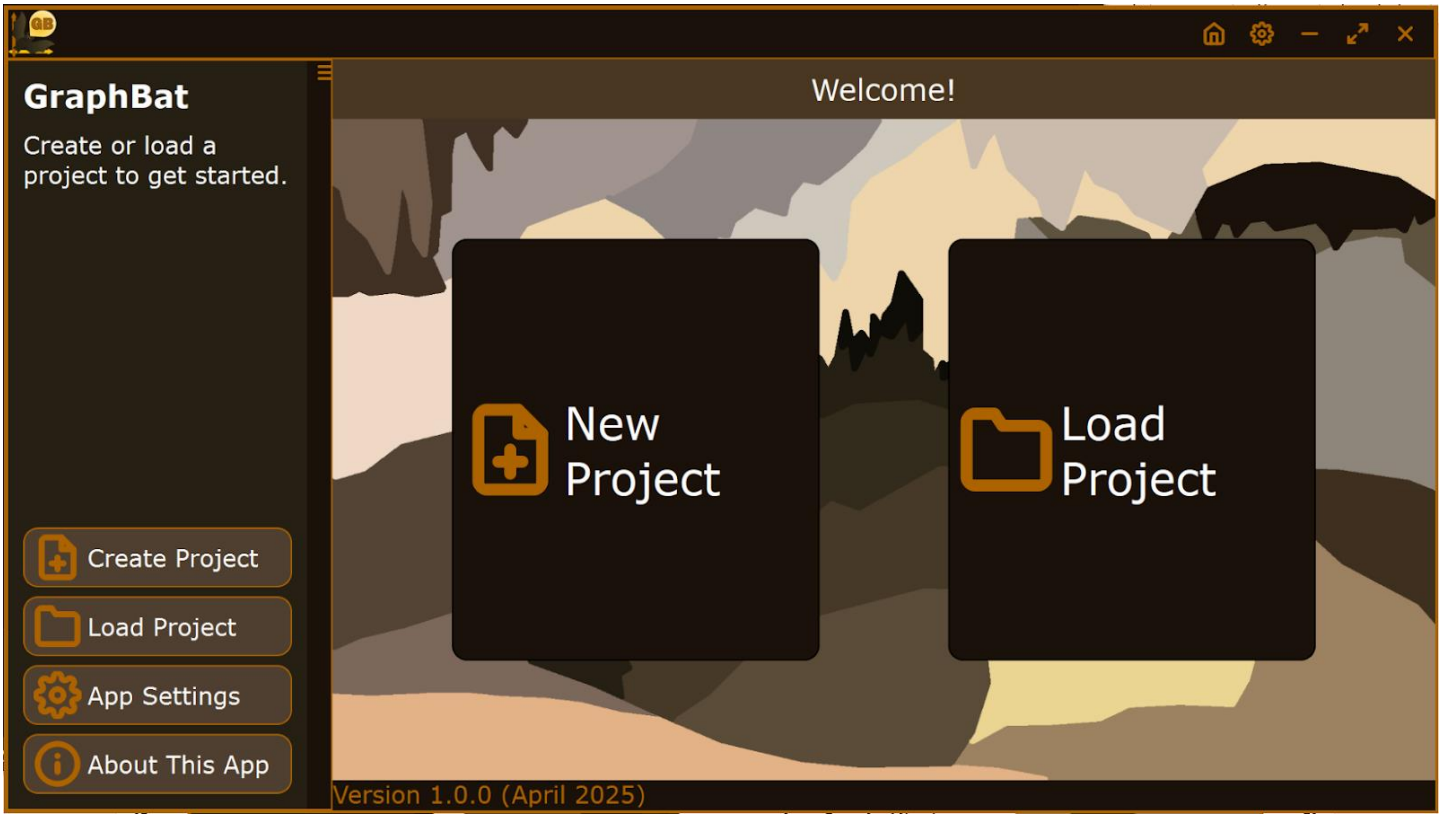

Figure 8: An implemented prototype of GraphBat using QT

Logo and name design was an important stage of design as having a concrete idea of the software project helped motivate our team to flesh out the features. Additionally, a unique and memorable name could make the final application easier for future users to locate, identify, and pass on to their peers.

The GraphBat logo was created using the primary colors from the application. It features two arrows symbolizing the axes of a graph and a bat.

Figure 9: The final GraphBat logo design

Accessibility

Since the nature of the application is to be a visualizer, it was critical to have accessibility options for those with visual disabilities, including but not limited to color blindness, low vision, and dyslexia. Compensating for these aspects was relatively simple to do with PyQt and PySide by modifying the application's stylesheets.

Color blindness may impact a user's ability to clearly discern between multiple lines on a line graph, grid tiles on the heat map visualization, and certain generic buttons. Using different line styles by default provided a secondary way to discern lines. Ensuring all buttons have a distinct icon or text likewise differentiated options. For grid tiles, allowing a user to customize what colors are used for the gradient allows users to choose the option that gives them the greatest accessibility.

Low vision may impact a user's ability to perceive any part of the application, particularly low-contrast text or small font. By default, the application was designed to provide high-contrast text (either fully white text on nearly black backgrounds, or black text on white backgrounds). However, font size proved to be a difficult aspect to dynamically change and was determined to be outside the scope of the current project.

Dyslexia may impact a user's ability to easily read graph labels and other parts of the application UI. Users are allowed to toggle on a dyslexic-friendly font in which to render all areas of the application UI.

Database Connection

Since the application is intended to run fully locally on a user's machine, there are no web connectivity aspects to the project. Therefore, the application does not connect to a remote database. All data pertaining to the application, such as datasets input by the user or created graphs, are stored within the application files. Since the data is not high-risk, there is no need for encryption or more intensive security measures.

Currently, the project is set to save user files in their current project folder, which can be easily located in the user's projects folder by name. The user's .CSV files are saved and copied multiple times during the process of creating a project and plotting data. The

user's raw .CSV files are copied upon project creation for backup and are saved at each step of processing, from initial cleaning to anomaly handling and interpolation of resulting missing data. This method guarantees that the original files are never modified, providing a reliable way to track and revert changes if needed.

Project Setup

The project was created in **Python version 3.13.0** using machines running **Windows 10** or later. To install and run the project's source code, the user will need to install Python. An IDE such as **Visual Studio** or **PyCharm** may be used to easily open, edit, or run the source code, but the code may be viewed with any text editor, such as Notepad, and run from the command line.

In addition, the user must install the following Python libraries on their machine:

- **Matplotlib** – used for generating graphs
- **PySide6** – used for some UI elements
- **PyQt5** – used for some UI elements
- **Numpy** – used for some mathematical computations
- **Mplcursors** – used for interactive graph elements

The following libraries were used for development, but are not needed to run the code:

- **PyQt5-tools** – used to develop some UI element packages

The below tools were used to develop the source code, but are not needed to open, editor, or view the source code:

- **Visual Studio/PyCharm** – Python IDEs
- **Qt Creator** – drag-and-drop Python UI editor

Test Plan and Report

Objectives

The software test plan described in this document outlines the project's approach to ensuring a high-quality, robust application for use in real-world settings by actual users. After completion of the test plan, the application should achieve the following:

- **Be easily used and understood** by a wide range of users, including those with average computer literacy or of minimal knowledge of speleology
- **Encounter minimal errors and zero critical errors** during general use of the application and processing of data
- **Recover safely from errors** and provide the user context for the errors
- **Pass a total of at least 90% of tests** performed on the application across all testing categories

Scope

Testing was conducted over the course of two weeks from March 31st to April 13th by all four team members. Each team member was responsible for testing and evaluating the area of the project that fell under their role's responsibilities. User testing involved external users not affiliated with the original project.

All features marked for testing below were tested and evaluated during the testing period. Once testing concluded, a summary of the results was provided in a software test report and attached to this document.

This project concluded April 28th, 2025. After this date, the project has no scheduled updates for additional testing or new features but may still receive updates later.

Test Items

Refer to the following sections for additional information related to the project. This STP will reference material from the sections below.

- Software Requirements Specification (SRS)
- Software Design Document (SDD)

Features Included in Testing

The application's features have been divided into phases of testing outlined below.

Project Creation

Test Case ID	Req. ID(s)	Feature Description	Pass / Fail
ImportDataset	1.1.1	Allows user to import zero or more datasets in . .CSV or .TXT format	Pass
ImportCavemap	1.1.2	Allows user to import zero or more cave maps in .JPG or .PNG format	Pass
ProjNameCount	1.2.1	Allows users to input a project name that is at most 40 characters	Pass
ProjNameType	1.2.2	Allows users to use alphanumeric characters, hyphens, underscores, and parentheses.	Pass
ProjNameCount-Alert	1.2.3	Alert user if their project exceeds 40 characters	Pass
ProjNameType-Alert	1.2.4	Alert user if invalid character has been used in project name	Pass
DuplicateName-Alert	1.2.5	Alert user if another project of the same name exists when attempting to save	Pass

ProjDescription-Count	1.3.1	Allows users to input a project description, at most 500 characters	Pass
ProjDescription-Alert	1.3.2	Alert users if length of description exceeds 500 characters	Pass
ProjDescription-Counter	1.3.3	Shows current description length, update when user edits	Fail
ProjNameDesc-Valid	1.4.1	Allows users to save project only if project name and description are valid	Pass
ProjNameDesc-Alert	1.4.3	Alert user if project name or description aren't valid	Pass
ProjNoData	1.4.5	Alert user if no dataset is attached when saving, have them confirm	Pass
ProjSaveState	1.4.6	Shall commit the project when user selects save option	Pass
ConfirmDiscard	1.5.1	Require users to confirm decision to discard their changes	Fail
ModifyProject	1.6.1	Allows users to modify an existing project's name and description	Pass
ModifyContent-Valid	1.6.2	Alert user if any changes made are not valid	Pass
ModifyContent-Confirm	1.6.4	Only commit changes once the user has confirmed them	Pass
ModifyContent-Discard	1.6.5	Allow user to discard changes	Pass
ProjDeleteConfirm	1.7.1	User must confirm deletion of project	Pass

ProjectDeleteAll	1.7.2	System shall remove all data related to deleted project once confirmed by user	Pass
ProjectList	1.8.1	System will display all created projects in order of creation	Pass
ProjectLoadAll	1.8.2	System shall load all data associated with selected project	Pass
ProjectLoadAny	1.8.3	System shall allow users to load a different project at any time	Pass
ProjectDisplay-Current	1.8.4	System shall display correct name and content of currently loaded project on all screen associated	Pass

Importing Data

Test Case ID	Req. ID(s)	Feature Description	Pass / Fail
DataTimeFormat	2.1.1	System shall import data and prompt users to select an index and data column	Pass
DataDetectFail	2.1.2	If imported file doesn't match any supported types, alert user of issue	Fail
DataStore- Import	2.1.4	System shall divide columns of data into Time and DataType, with the DataType being the only column and Time being the index and store as copy	Pass
DataNameLabel	2.1.5	System shall allow user to enter a label per sensor of at least one character and at most ten characters	Pass

DataNameAlert	2.1.6	System shall alert user if the illegal characters are used or if the name is duplicated	Pass
DataRemoveColumn	2.1.7	System shall allow user to remove a column of data after it is imported	Fail
DataDetect-Anomaly	2.2.1	System shall detect anomalies in data based on IQR and spike detection	Pass
DataAnomaly- Alert	2.2.2	Shall alert list of anomalies and give option to remove, ignore, or view	Pass
RemoveAnomaly	2.2.3	Shall remove all selected anomalies and properly interpolate values	Pass
DataKeepCopy	2.1.3	System will made removals and edits on the copied file only, not original	Pass

Graph Visualizations

Test Case ID	Req. ID(s)	Feature Description	Pass / Fail
ProjSave-Visualization	3.1	System shall properly save the current visualizations as a picture file	Pass
ProjLoad-Visualization	3.2.1	System shall properly load saved visualization(s) and display on screen	Fail
ProjDelVis-Confirm	3.3.1	System shall delete visualization after user confirmation	Fail
GraphSelect-Range	3.3.1	Allow user to select time range of data to be graphed	Pass

GraphXAxisValid	3.3.3	Require valid column type for the X axis	Pass
GraphYAxisValid	3.3.4	Require valid column type for the Y axis	Pass
GraphDraw-Verified	3.3.5	Draw graph if data and axes have valid columns selected	Pass
GraphDisplayAll	3.3.6	Graph will draw a continuous line between data points	Pass
GraphDisplay-Axes	3.3.7	Graph shall display correct labels	Pass
GraphSelect-Point	3.3.8	System shall display the x and y intersection value for each point when hovered over	Fail
GraphUpdate	3.3.9	Update and redraw graph whenever user modifies an element	Pass
GraphNoData-Alert	3.3.10	Alert user to attach dataset if none exists	Fail
GraphAxisRange	3.3.11	System shall denote each axis with 9 tick marks, each marking 10% of data's range	Pass
LineSelectPoint	3.4.2.	System shall display the x and y intersection value for each point when hovered over	Pass
LineShowAll	3.4.2	Graph will show a continuous line through all points displayed regardless of number of sensors displayed or anomaly dialog choice	Pass
Histogram-SelectBin	3.6.1	System shall display the x and y intersection value for each bin when hovered over	Fail
HistogramXBin	3.6.3	System will divide data into bins into plot them on x-axis	Pass

HistogramYBin	3.6.4	System will calculate the frequencies of each bin and plot them on y-axis	Pass
BarType	3.7.1	Allow user to choose type of aggregated data (min/max, average, etc.)	Pass
BarSelect	3.7.2	System shall display the x and y intersection value for each bar when hovered over	Pass
BoxSelect	3.8.1	System shall display the quartile values for each box when hovered over	Fail

Heat Map Visualization

Test Case ID	Req. ID(s)	Feature Description	Pass / Fail
CaveNoData	4.1.1	System shall prompt user to attach a dataset to the project if none exist	Fail
CaveNoMap	4.1.2	System shall prompt user to attach a cave map to the project if none exists	Pass
CaveGenerateGrid	4.1.3	System shall generate a grid of tiles based on image	Pass
CaveDrawMask	4.1.4	Allow user to draw a selection to include in their current render	Pass
CaveHideOuter	4.1.5	System shall hide all grid tiles that lie entirely outside the selection area	Pass

CaveChooseTile	4.1.6	Allow user to select tile that represents the collection location of each dataset	Pass
CaveCalculateDist	4.1.7	System shall calculate the value for each tile based on its distance from the each sensor tile	Pass
CaveSelectPoint	4.1.8	System shall display the value of a tile when it is hovered over	Pass
CaveAutoColor	4.1.9	System shall automatically use the colors purple and pink as the low and high ends of data range	Pass
CaveChooseDev	4.1.10	Allows users to choose how many standard deviations to plot	Pass
CaveColorMap	4.1.11	System shall color each tile based on its distance and the chosen color	Pass

Exporting Data

Test Case ID	Req. ID(s)	Feature Description	Pass / Fail
ExportVisualization	5.1	System will export the current visualization to the user's selected directory path	Pass
ExportMapVideo	5.2	System will export the current heat map visualization as a video (.mp4)	Pass
ExportMapSnippet	5.3	System will export the current visualization timestamp aggregate as an image (.png)	Pass

Application Settings

Test Case ID	Req. ID(s)	Feature Description	Pass / Fail
SelectTheme	6.1	System will allow user to select from a list of available themes	Pass
SelectAppFont	6.2	System will allow user to select from a list of application fonts to display	Pass
SelectGraphFont	6.3	System will allow user to select from a list of graph fonts to display	Fail
ChangeWindowSize	6.4	System will allow user to minimize or maximize window while retaining formatting	Fail
ScaleWindowSize	6.5	System will allow user to shrink or enlarge window while retaining formatting	Fail

Features Excluded From Testing

Feature Description	Reasoning
1.4.2: Saving new project whether or not there is a dataset attached	Feature is realized by default
1.4.7: Require user to confirm their decision to save twice if project doesn't have dataset	Should minimize steps needed for project creation, must add dataset before visualizations anyways
2.1.9: System shall remember the last user-defined column typing and ordering formatting and will use it as default	Should have user explicitly name data type in case of errors in automatically trying a similar type

3.4.8: The system shall add every nth point to the graph with a default n of 50.	Better to graph every point in the dataset within limit
3.4.9: System shall draw a line from left to right between each point consecutively	Reworded in order to state the requirements for no holes in a continuous graph

Approach

Unit Testing

The first stage of testing consisted of using pytest to perform checks on many of the requirements listed above. Most of these are individual functions and pytest is able to check if each one passes or fails given various inputs.

Integration Testing

Integration testing was performed with tools like pytest, tracemalloc, line_profiler, objgraph, and cProfile. Here, we began testing the overall performance of the units once they were integrated together.

Interface Testing

Interface testing began once integration testing was complete. Here we ensured the UI and backend communicated properly and that the various functionalities were implemented and functional.

User Testing

This was the final stage of testing. A series of users external to the project were recruited for user testing. User testers were uncompensated due to the project's fixed budget of $0. The testers were given access to the semi-final application and user manual and tasked with accomplishing a series of objectives relying solely on their own knowledge and the user manual.

Each user was monitored during testing by at least one team member. The team member noted the user's reactions to tasks, how much difficulty they appeared to have with accomplishing tasks, whether the user succeeded in each task, and any additional comments the user made. The results of this stage of testing were used to adjust design and performance of the application.

Environmental Requirements

Hardware

- **Network Requirements:** The application and all testing tools work fully offline, so there are no minimal network requirements to test the application.
- **RAM:** At least 4 GB of RAM are required to run visualizations with small datasets; for larger datasets, 8 GB of RAM may be required.
- **Memory:** Users should expect to reserve at least 10 GB of hard-drive memory to ensure sufficient storage space is available for the application, datasets, and any additional files generated during operation. The largest file size would be the rendered cave heatmaps, which could be multiple GBs each.

Software

The application and all associated tests were written in Python and can be run from any IDE that supports Python development, such as Pycharm, or from the command line. No special software is required to run the application or its tests.

Tools

In order to test the speed and memory usage of the program, tools such as pytest, tracemalloc, line_profiler, objgraph, and cProfile were used. The heatmap visualizer is very memory intensive and computationally expensive. These tools were useful in improving performance by showing bottlenecks and any potential memory leaks. The pytest framework was used to test many of the more boolean-like test cases with its assert statement.

Version Control

Our team utilized Github as the primary version control system, complemented by Discord for team communication. This dual platform approach allowed us to maintain structured code repositories while facilitating real-time discussion about code changes. We leveraged Github to maintain a centralized code repository to track all code modifications through commits and to manage parallel development via branching. Throughout project milestones, we actively communicated on sections of the project that each member was responsible for in order to ensure little overlap in individual tasks and overall efficiency.

Conclusion

The purpose of this project was to develop a tool specifically designed for speleologists to visualize data such as temperature, humidity, and more in subterranean environments with complex geometry. Comparable tools, like ArcGIS, require extra effort on the user's part to get the same results. GraphBat is an open-sourced application. This choice was made to assist scientists—as they often have minimal funding—and to allow them the freedom to adjust the source code to better suit their needs. Development followed an incremental approach with changes made as the scope expanded to encompass various issues and unforeseen needs. The project, as of April 29th, 2025, is roughly 90% complete. There are lingering cosmetic bugs with rendering the GUI on some devices and a few data handling issues. The heatmap, in particular, needs additional optimization. Despite these issues, GraphBat is a functional app that can produce the visuals it was intended for. In the future, after some final fixes, we hope to present GraphBat to the National Speleological Society and other caving organizations to promote the open-source tool to those who may benefit from it, and to gather input and advice to further develop the application.

Appendix

SCCi Guidelines

The **Southeastern Cave Conservancy** allowed us to place sensors in one of their cave preserves. To do this, we submitted a **Research Permit Application** and two **Visitation Permits** on the dates we visited to place and pick up the sensors.

The SCCi is dedicated to preserving caves and ensuring visitors know how to cave safely. For these issues, the SCCi mirrors the **National Speleological Society's** guidelines. These include informing visitors of the various dangers like falling, hypothermia, falling objects, and getting lost or trapped. With proper preparation, education, and awareness, most accidents can be avoided. Preparation includes proper gear, setting up an emergency contact, checking the weather, and more. Education includes understanding the conditions and dangers inside of caves, how to avoid them, and how to deal with any issues that may arise.

To preserve caves, visitors are to follow all laws regarding caves and follow a **"leave no trace"** principle. This means that upon exiting a cave, the cave should have no trace of one's visit. Of course, exceptions are made for things like research, which is why that requires a special permit. The SCCi makes each visitor sign a permit that states that they have read and will follow their guidelines.

Due to the sensitive nature of caves, cave locations and maps are not to be shared openly and should only be given out by those in charge of the cave.

Please refer to the SCCi website for any additional info: https://saveyourcaves.org/

Risk Management

The risk management plan below was developed during early project conception and adhered to over the course of the project. The most likely causes of project failure were identified and described, and a plan was put in place to minimize each risk.

Time Miscalculation—Priority: High

Description: May occur as a result of a mis-estimation of the time required for a task, causing the overall testing process to be skewed. Continued time miscalculations may lead to cascading problems. Careful monitoring of test progress is critical for success.

Mitigation Plan:

- Subdivide high-level testing stages into atomic tests
- Review and update progress regularly, adjusting as needed
- Generously estimate time required for the most important testing stages

Skill Failure—Priority: Medium

Description: May occur as a result of a team member being unable to adequately perform or complete a task due to lacking the necessary skill; mitigation is especially difficult due to unavoidable constraints on available resources.

Mitigation Plan:

- Discuss each team member's skill sets ahead of commencing testing
- Check in with team members to assess their progress and provide assistance
- Reassign tasks as needed if a team member is struggling to complete it alone

Team Miscommunication—Priority: Low

Description: May occur due to inadequately explained requirements, project scheduling, or other parts of the project; may also occur as a result of poor communication between team members regarding personal schedules.

Mitigation Plan:

- Maintain continuous, timely communication between team members
- Schedule all-hands meetings to coordinate team efforts
- Require all team members to thoroughly review project documents and communicate concerns or confusion as they arise

References

[1] D. Zhang and I. Lee, "Interpolation of Sensory Data in the Presence of Obstacles," *Procedia Computer Science*, vol. 29, pp. 2496–2506, Jun. 2014, doi: https://doi.org/10.1016/j.procs.2014.05.233.

[2] Wikipedia Contributors, "Inverse distance weighting," *Wikipedia*, Mar. 04, 2019. https://en.wikipedia.org/wiki/Inverse_distance_weighting

[3] Wikipedia Contributors, "Natural-neighbor interpolation," *Wikipedia*, Aug. 19, 2024. https://en.wikipedia.org/wiki/Natural-neighbor_interpolation

[4] A. T. DeGaetano and B. N. Belcher, "Spatial Interpolation of Daily Maximum and Minimum Air Temperature Based on Meteorological Model Analyses and Independent Observations," *Journal of Applied Meteorology and Climatology*, vol. 46, no. 11, pp. 1981–1992, Nov. 2007, doi: https://doi.org/10.1175/2007JAMC1536.1.

[5] PythonGUIs, "Python GUIs: Create GUI Applications with Python," PythonGUIs.com, [Online]. Available: https://www.pythonguis.com/. [Accessed: February 21, 2025].

[6] Knott, Ray. "Cave Safely." www.saveyourcaves.org, www.saveyourcaves.org/learn/cave-safely.html. [Accessed: April 20, 2025].