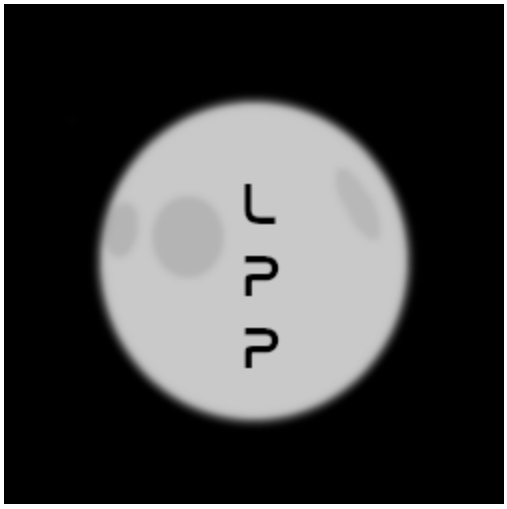


Requirements Documentation

April 18th, 2024

The Lunar Pit Patrol:

Evan Palmisano, Ibraham Hmood, Alden Smith, Caden Tedeschi, Levi Watlington

Project Sponsor: Trent Hare

Faculty Mentor: Vahid Nikoonejad Fard

Version 1.1

Introduction

Our project is a graphical user interface for displaying crater-related statistics. It interfaces with a command-line application which returns data such as the number of craters in a region. This information can be very beneficial for the space industry, as it can be used for mapping celestial surfaces and figuring out the ideal spot to land spacecraft on other planets. The space industry is worth 1.8 trillion dollars, with some companies worth hundreds of billions. Also, there is a lot of unpredictability when it comes to landing spacecraft, and it is necessary to have as much data as possible about the area we are landing in, and our project assists in that. Our sponsor (Trent Hare) and his associate (Marc Hunter) have developed the command-line application on which our project relies.

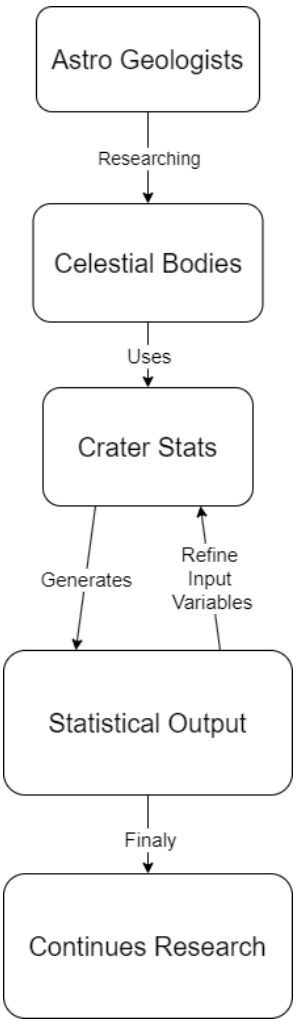
Since the program currently relies on a command-line interface, it can be difficult to interpret the data. Developing a graphical user interface can make data interpretation easier. Astrogeologists, who are most likely to use this project, are not trained to understand the output of a command-line application. Thus, a GUI makes interpreting data easier.

As stated before, our sponsor and their associate have developed the command-line application that our project relies on. They work for the USGS, so they receive funding from the federal government for their project. Concerning the command-line program, our goal is to interface it with a GUI that is consistent and easy to use. Our solution is to make an interface that plots the statistical data provided by the program on graphs.

As stated before, this project will be used by the space industry. Not only can it be used for landing objects on planetary surfaces, but this same project can be used to determine the age of a planet's surface. This project can also be used on non-planetary bodies, such as moons.

Problem Statement

The problem is using crater stats is too difficult. Currently, the Crater Stats application is just a command line interface (CLI), and it is difficult for those not used to CLI to remember all the possible commands to get exactly what you want. Then, if it needs to be refined to get a slightly different graph, it has to be done all over again. “Our scientists should not have to go through a boot camp to use Crater Stats…” - T. Hare. An attempt has already been made to create a user-friendly experience by creating a Graphical User Interface (GUI) for the Crater stats application. The GUI was a single page with everything presented all at once. The problem was it lacked a user experience design and was difficult to use. We will fix the usability problems Crater Stats faces by researching and rapidly prototyping smarter user experiences that reduce frustration and time spent gathering accurate statistics. Additionally, it will be beneficial to display the CLI command that would generate the graph shown through the GUI.



The USGS Astro Geologists use Crater Stats to study and research various statistics about craters and how they shape celestial bodies. First, an Astro Geologist has a research project that needs statistical data. The researcher uses Crater Stats to generate useful graphical data. The researcher refines their input until they get exactly what they want. Then the researcher can use their gathered data. The slowdown is the generation and refinement phase, our primary target to remedy.

The problem is that Crater Stats needs a GUI with user experience in mind. Currently, the application is too difficult for a smooth and fast experience. The application must be self-explanatory which we will remedy by simplifying the GUI. The application is currently time inefficient with CLI and must be faster through an easy-to-use GUI. The best way to produce the best product is to prototype and test solutions rapidly. This way we ensure Crater Stats is a more useful product overall.

Solution Vision

Now that we've discussed the current limitations of being command-line only, let's get into our solution. We aim to create a graphical user interface (GUI) for the current Craterstats program to allow those in the astro-geological community who are not as well-versed in the command line to use the program easily. This new interface will be built off of the old version used before becoming outdated and being moved to the command line. Some updates will be made to our project like adding more spacing, making sure the program has better readability, relocating features to different pages, and the ability to input a command line argument to create a graph. If time allows, we will also implement the GUI on a website creating easier access to the program. These changes will help make this new interface better than it previously has been:

- *Extra padding between items* - Padding, the space between items, was limited in the oldest version of the software. Adding extra padding to the items will allow for better readability between items with less chance of making an accidental change to any part of the plot.
- *Relocating features to different pages* - In the old software every setting or function of the program was put onto a single page, in our new GUI we are planning on having three separate pages for each of the major areas of the program; Global Settings, Plot Settings, and Plot. Adding these three pages will allow items to be spread out with less information all in one place.
- *Ability to input a command line argument* - The ability to input a command line argument will let those using the current version keep using it like they are used to. This feature will also display what the command line version of the setting that has been inputted would be.

Our solution will turn the data that the user inputs, things like a chronology function, production function, plot style, and plot symbols, into a plot that the user can interpret and use. Our main process will be using the logic and algorithms implemented in the current software. Creating this GUI will add ease of use to the program for anyone without knowledge of the command line. A tradeoff this possibly brings is the speed at which a plot can be created. With multiple pages and setting options taking time to

input, typing out a single line in the command line will be faster for those with the knowledge of the arguments needed.

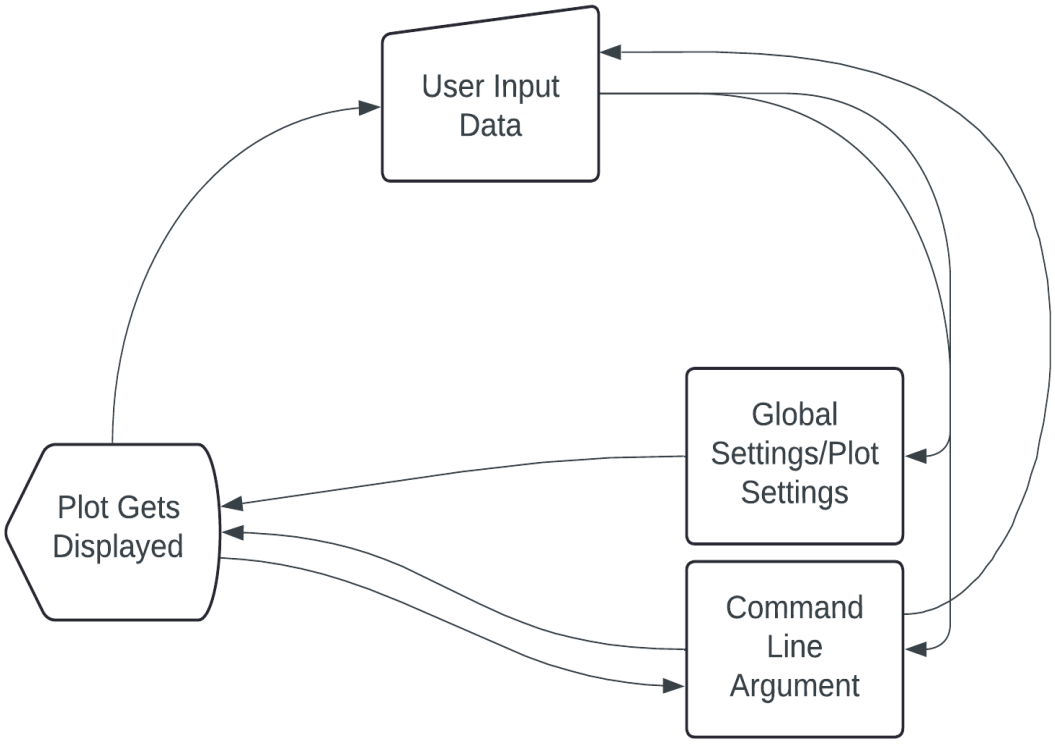


Figure 1. Improved program workflow

Project Requirements

Our domain-level requirements focus on setting up a GUI and managing the different graph settings so our users can have the best and easiest experience possible. Because of the nature of our Capstone project, we don't have too many domain-level requirements since we are only making a GUI application for an already existing CLI application. Keeping the goal of our Capstone in mind, our domain-level requirements are as follows:

1. Easy to use by regular people that don’t know how to use the command line
2. Shows the correct graph and information when used
3. Accepts graphs already correctly made and allows saving of user-made graphs

4. Ability to use without installing everything

Functional Requirements

As mentioned earlier, we aren't making an entire program; we are just wrapping one in a GUI application for easy use by the average person, so we don't have too many functional requirements other than getting the right information from the CLI application and displaying it for the user to see. The following functional requirements are separated into four parts based on which domain-level requirement they are for.

Domain-level requirement 1 functional requirements:

- Buttons and dropdowns with obvious labels that modify the graph
- Section off the different types of settings so users know how each option will modify the graph
    - Create multiple frames for our GUI to give it enough space to section off the different types of settings
- Easy navigation through the frames of the GUI
    - Create tabs for each frame so the user doesn’t have to waste time navigating through each frame using buttons

Domain-level requirement 2 functional requirements:

- Clicking any button or drop-down menu will update the graph
    - Clicking any button or drop-down menu will trigger a function that redisplays the graph based on the button or drop-down pushed
- Clicking any button or drop-down menu will update the command line command used to make the graph
    - Clicking any button or drop-down menu will trigger a function that redisplays the command line command based on the button or drop-down pushed
- Have understandable tab labels

Domain-level requirement 3 functional requirements:

- Allow users to select files with correct graph settings to open them in our application to display the graph they want along with the settings on the GUI filled in
    - Create a function that allows for the selection of a specific type of file to get graph information from
    - Create a function that allows for the current graph information to be saved in a file

Domain-level requirement 4 functional requirements:

- Allow users to use our GUI application through a website
    - Set up a server to run by itself
    - Integrate the Dear PyGUI application into the website for users to use online

Performance Requirements

Because our project just calls functions from the Craterstats CLI app to create the graphs, we don't have any computation-heavy functions in our GUI app. The major obstacle our project will tackle is to check which buttons and dropdown options are selected so it can have the correct graph displayed. Our project's largest computation isn't very memory intensive either since it isn't checking enough buttons and drop-down menus to have the check take more than a second.

Since our project calls the CLI app functions to get the graphs, we can ensure the accuracy of the graphs displayed. We then plug the information from the settings chosen into the graph creation functions for generation. There is no chance of our display being wrong unless the user inputs the incorrect setting for the graph they want since the graph functions just take the values from the settings options and plug them in. We ensure our buttons, drop-down menus, and tab buttons are accurate too since they give the correct information to our application and, in the tabs case, take the user to the correct frame when pushed.

The responsiveness of our GUI is very good because our application isn't creating the graphs itself, the CLI app is, so all we need to do is check every button and drop down to find the values that need to be plugged into the graph function. After the function returns the graph all we need to do is replace the previous one in the GUI with it to keep the graph updated. Also, when any of the buttons or dropdowns are pressed the graph will be updated right after and when the tabs are pressed the user gets taken to the frame that they clicked almost immediately.

It is very easy to learn how to use our GUI because one of the main points of our project is to give regular people with no experience using the command line an easy way to get the graphs they need. The only things users need to know to use our GUI are the different types of information that the graphs will display, so they know what each button will do, and how to use a mouse. Everything other than inputting the title of the graph and the downloading and uploading of graph information will be handled by the GUI. The user has to do as little as possible to get the information they want.

Due to the ease of use that our project requires, the only training that users will need to use our GUI application is in the knowledge that the graphs we display represent, because if the user doesn't know what any of the information on the graph is then they won't know what settings they need for. Since there isn't a lot of training needed to use our GUI we suspect there won't be much of a difference between novice and experienced users. There is nothing mentally taxing in our application so even if a person uses it repeatedly the only way they will get better at using it would be to memorize where every option is so it can be chosen faster.

Environmental Requirements

Our project is just the creation of the GUI application. The environment needed for our project needs to have everything the original CraterStats CLI application had. All the packages, libraries, and other files imports take up 1.44GB of space on a system's hard drive. Our GUI application is only a few MB so the user should at least have 1.6GB of space available on their hard drive to download everything.

One of our sponsor's environmental requirements was that whatever frameworks and languages we used for this project needed to be easily maintainable. Since we

need to keep maintainability in mind we chose to do everything except the website portion of the project in Python since that is what the original CraterStats CLI application was written in. Since we wrote our project in Python instead of C as we were thinking about, we don't need to do any extra coding to make the Python and C portions of the application work well together.

The next environmental requirement for our application was that it needs to be able to be made into a web application. The web application needs to be able to do everything that the GUI application can so we need to download the 1.6GB of stuff for the CraterStats CLI application onto the server so it has access to all of the CraterStats functions and graphs. The website needs to be maintainable as well so when we do anything big with graphics for our website using THREEjs we comment on what is going on so anyone can read them and follow along with what the code is doing and know what each portion of it will affect.

The last environmental requirement is to be able to download and upload files that hold specific settings for a graph to and from our GUI application. Our application needs to allow the users to choose what file they want to upload to the application from their file manager. When they download the current graph settings to their system, they need to be able to name the file whatever they want. When they download the file, it will be placed in a directory for downloaded files.

Potential Risks

There are some potential risks associated with our graphical user interface. One potential risk is inaccurate data being displayed. Another potential risk is data being displayed incorrectly in such a way that results in inaccurate interpretations. Some more potential risks involve how the GUI is programmed and designed. If our GUI is not properly programmed and designed, then that can lead to performance issues. If our GUI is not properly designed, then our users could end up confused or not know how to use it properly. One more potential risk is a competitor releasing a program with a more intuitive and easy-to-use interface. All of these potential risks have their outcomes, some of which are similar. Concerning inaccurate data being displayed, users could

come to false conclusions. Since scientists will use this interface, inaccurate data being displayed could result in them forming incorrect theories.

Then concerning data being displayed incorrectly, the outcome could end up being the same. That is to say that our users could end up forming incorrect conclusions based on the incorrectly displayed data. Since the people using our interface are likely to be scientists, this could result in them forming incorrect theories. Potential issues arise not just with how data is displayed or how accurate the data is, but also with an improperly designed and programmed interface. Such an interface could result in performance issues, which could drive our users to alternative programs. An interface that is improperly designed could also result in our users being confused, which could drive them to use a program that has a more intuitive UI.

Lastly, a competitor could end up releasing a product with a more intuitive and easy-to-use interface. If that is the case, then we could end up losing users to that program. With all of these risks, there are ways for us to adapt. For one, we could introduce our designs to a sample set of users to ensure that it is easy to use. We can also rigorously test our data display libraries to ensure that they accurately display data. We can also try and minimize the activity on our program's main thread, maybe even use more threads wherever needed. All in all, there are risks associated with our graphical user interface. However, we can adapt and mitigate the impact of these risks early on during development.

Project Plan

So far, we have established a working prototype that consists of a welcome, format, statistics, and a plot view page. So from this point, where do we go from here? We are on the cusp of completing our prototype using DearPyGUI. To progress in the right direction, we need to plan the sequence of our considered future implementations.

Our first future milestone is to implement necessary buttons, drop-down menus, and other forms of input. We don't need to implement every single option for input on the first milestone, but if we could integrate some necessary options such as loading files, padding our available options for ease of navigation, tabbed windows, and

selecting the type of view for a specified plot. We hope to achieve this by the end of the current spring 2024 semester.

The second milestone is going to be transitioning from our domain lrv 1 requirements to level 2. This milestone focuses more on the functional usage of the application. We need every available option from domain level 1 to be functional so that the application reacts accordingly once input is received. The graph should update immediately after the option is selected. This is something that would be nice to have implemented during the summer by the time the fall semester starts.

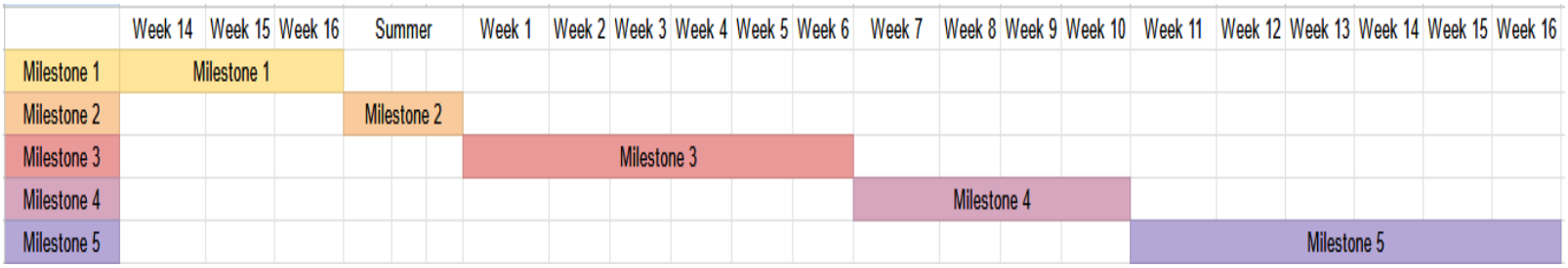
The third Milestone is to implement a feature in the GUI that can generate a command that would be used in the command line version application. This will help improve familiarity with the application and improve its user experience. A cool implementation to build on that is if the program could take input from both the command line and GUI simultaneously. A good goal would be to have this by week 6 of the Fall semester.

Our fourth milestone right now is to implement our domain level 3 requirements. This milestone focuses on file processing and functionality. One update is to give users the ability to select files and have the application pre-load settings display the plot immediately. Users will also be able to select specific file types. This should be added by week 10 of the Fall semester.

Our fifth and final milestone is to develop a web version of the application and host it on a server to increase our total number of users and improve accessibility. This would be worked on for the remainder of the semester up until week 15.

Overall, the progression of this project has a defined set of goals and objectives, however, this could be subject to change. A project in software development can be dynamic and the team will have to react accordingly. This set of milestones will help keep us on track and guide us as we continue to nurture this project.

GANTT Chart of team milestones



Conclusion

In summary, the Lunar Pit Patrol has a clear understanding of the upcoming phases of the Crater Stats capstone project. Switching our library utilization from Tkinter to DearPyGUI will accelerate the time it takes for the team to make our implementations. DearPy presents itself as an easier library to work with for developing the application. Its function purposes are more aligned with the team's requirements and it alleviates certain issues encountered with Tkinter such as conflicting default themes based on the user's operating system.

Using DearPy, our current short-term implementations include relocating functions and features to their respective pages. This will increase the ease of development for the team when implementing features regarding our first milestone. Our first milestone features will improve the navigation experience and ease of use for users utilizing this application for their studies. This milestone should be completed by the end of this current semester (Spring 2024).

Taking into consideration our functional, physical, and environmental requirements, our application's development will need to be monitored. Our team will be assessing the performance of the application after the first milestone is achieved. Not only do we want our application to work but we want it to be responsive and enjoyable to use. In turn, we will be executing our additions concerning our requirements and potential risks of the application.