Tech Feasibility

March 22nd, 2024

The Lunar Pit Patrol:

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

Project Sponsor: Trent Hare

Faculty Mentor: Vahid Nikoonejad Fard

Introduction

Commercialized celestial landings are becoming increasingly prevalent in modern-day society as the race to colonize the Moon and Mars is growing rapidly. The main feature of these missions at the moment is autonomous, unmanned landings executed by specialized vehicles. These vehicles need to land appropriately to the mission plan as enormous amounts of funding are directed towards these missions. The problem is that there is a lot of unpredictability when it comes to these landings and it is necessary to possess as much information about the landing surface as possible.

The current solution is the usage of crater statistics within astrogeology to map celestial surfaces and find ideal landing locations for these missions. Our sponsor, Trent Hare along with his associate Marc Hunter use a command line application developed in Python to interpret and assess given data files. The main problem our sponsor is facing is the fact that the current application operates through command-line interaction. This can pose a problem as the data output is not easily interpreted. As for astrogeologists, not everyone knows how to operate within the command line terminal which introduces a problem of usability.

The Lunar Pit Patrol's job is to modify the current application and develop a graphic user interface (GUI) so the application can operate on its own instead of the command line interface. This will improve readability and data assessment for astrogeologists which will promote success in their work. Some key features of the project include a standalone window for containing the application, a toolbar to access functions and modify statistics, an area for displaying plots of the relevant statistics, and immediate options for more common modifications of data and functions. These additional features and improvements to the application will ensure consistency, improve assessment times, and introduce ease of use.

Tech Challenges

There are a few challenges in our project that we have encountered and we are working on figuring out how we can solve them if we haven't already. One of the first major challenges we thought of is how we must connect the GUI CraterStats application to the command line application so that the GUI can send commands to the CLI and get the data it needs back from it. After searching through libraries we found the Popen and PIPE functions in the subprocess library which allow us to make a subprocess in our Tkinter GUI that can take text and pipe them to the CLI to run them as commands and then pipe the data the command gave back to the GUI to be displayed to the user. A separate solution to connecting these two applications is to build upon the existing application through a fork and strip the command line with buttons and switches.

Another major challenge is we will need to connect an entire Python application to be run within a web environment. We are researching if AWS Lambda Functions could potentially be our solution to run a Python backend with a web frontend. We will need the front end to closely mimic the likeness of the Python GUI application to maintain consistency in usage. Forking the current CLI application will maintain the old application to be better used with AWS Lambda Functions.

A lesser challenge would be how we need to best display the data given back from Crater Stats. There are several libraries that we can use when we enter this challenge such as py_graph or EasyGraph. However, the goal is to clearly display important data in such a way that is easily readable to the user. Being able to export or objectify these graphs will assist in displaying them on our later web application as well.

Technology Analysis

For this project to be successful, our goal is to simplify the use of craterstats while keeping the overall program the same. At the moment the craterstats program is only run out of the command line using a long and complicated argument system. This new GUI will need to perform the exact same functions as the command line system but in a way that is easy for users to understand. While keeping this same program it is important to highlight a few things from the command line system. When the user selects all the options for the graph they would like, a command line like statement should be shown that displays how the command would look when run on the command line. The program should also display the graph to be exported after the commands are run.

Desired Characteristics

Our ideal solution for this project is to have a user-friendly interface that is easy to use while also being easy to maintain after updates to the program. Having a user-friendly interface will allow access to anyone who would like to use the program. Having discernable commands that appear when needed will help keep the interface clutter free and easy to read and understand. Having an easily readable interface will allow for use of the program commands without needing to know the actual command line arguments needed and the order they are needed in. To help the ease of updating the GUI will need to be easily maintained. This will allow for easy updates to the GUI if there are updates to the program that change how it is used. The program will also need to be able to be used across platforms on Linux, Mac, and PC.

Alternatives

There are a lot of ways to fix this issue. Given that there are a lot of tools to choose from to be able to create a GUI there are quite a few options for a fix.

- Unity: Brought up because it is used by a few people in the group. Unity's editor allows for fast UI creation with ability to easily grab and edit different parts of the interface. Originally founded in 2004 by David Helgason, Nicholas Francis, and Joachim Ante, Unity was originally created as a Mac OS X game engine. However through the years it has grown to be used for many different uses including game development for multiple platforms, animation, and 3D modeling. [3]
- Windows Forms: This application was made known through research and previous use. Released in 2002 by the .NET foundation, Windows Forms (WinForms) was released as an early and easy way for developers to create graphical interfaces. Each major update after the initial release came with the newer .NET Framework versions, each making Windows Forms easier and faster to use. [4]
- TKinter: A library in python that was brought up by team members and a suggestion by our sponsor, was released as a GUI extension for the Tcl scripting language by John Ousterhout. Not only a library in python but also other extensions in Perl, Ada (Known as TASH), Ruby, and Common Lisp due to its popularity. Tkinter allows for easy creation of graphical user interfaces with widgets for nearly all

necessary functions for an interface. [1]

- DearPyGUI: Brought to our attention by our sponsor, the full version initially released in 2021 by Jonathon Hoffstadt. DearPyGUI is a python library that is used to create graphical user interfaces that also has tools for dynamic charts, tables, and drawings as well as tools for application development. [2]
- ReactJS: Suggested by our sponsor to have a web version of the program, the team looked into ReactJS for its small learning curve, and ease of use. Released in 2013 and developed by Meta, ReactJS (React) is a library of JavaScript that allows for better dynamically created user interface elements in a webpage.
- ThreeJS: Used previously by a team member, ThreeJS is a library of JavaScript that was released in 2010 by Ricardo Cabello. This library allows for in depth 3-Dimensional designs and animations to be used on a webpage using WebGL.

Analysis

When testing the above applications and libraries it became clear quickly which ones would be best for further pursuit. Windows Forms was immediately disconsidered because it is not cross platform compatible between Linux and Mac, as well as being harder to maintain. Unity, while being cross platform compatible, had too much overhead in the creation process with lots of additional dependencies required to create a build of the interface. It would also be nearly unmaintainable within our industry due to the excessive overhead requirements. TKinter and DearPyGUI are both already in python, so it is easier to expand upon the already existing craterstats program that is in python. TKinter was tested by creating simple programs to see how the library works. TKinter allows for easy placement of different functions and widgets with easy accessibility to each of the widgets, with access to creating different pages and sections of each page. This lets us as the designers make a user-friendly interface while keeping the overall program very manageable and maintainable for future updates. DearPyGUI is nearly identical to TKinter only with newer features and more capabilities. DearPyGUI being a python library gives the same easy expansion of the existing command line program as TKinter does. Installing and running DearPyGUI and Tkinter is as easy as doing a quick *pip install* and importing the library into a python application. ReactJS and ThreeJs are very popular and easy to learn libraries within JavaScript. Using these libraries to create a website edition of the graphical user interface, will create easier and faster accessibility for those that want to use the program. While creating simple programs and looking into different projects created with these two libraries showed just how effective they can be at creating a user-friendly experience.

Chosen Approach

Our chosen approach is to continue with the project within the TKinter library and ReactJS. While Windows Forms and Unity would have been easier to use and create an interface, they had too many problems with trying to get the craterstats program linked in with at as well as too much overhead for creation. DearPyGUI is another option if TKinter doesn’t work out, but it was introduced to us a bit later so there hasn’t been much research or testing with it. ReactJS was also chosen because our sponsor wants both a standalone program as well as a website.

	User Friendly	Ease of Maintenance	Cross Platform Compatibility	Total
Unity/	4/5	1/5	5/5	10/15
Windows Forms	4/5	1/5	0/5	5/15
TKinter	3/5	5/5	5/5	13/15
DearPyGUI	4/5	5/5	5/5	14/15
ReactJS	3/5	4/5	5/5	12/15
ThreeJS	3/5	3/5	5/5	11/15

Table A: Comparison between GUI creating environments

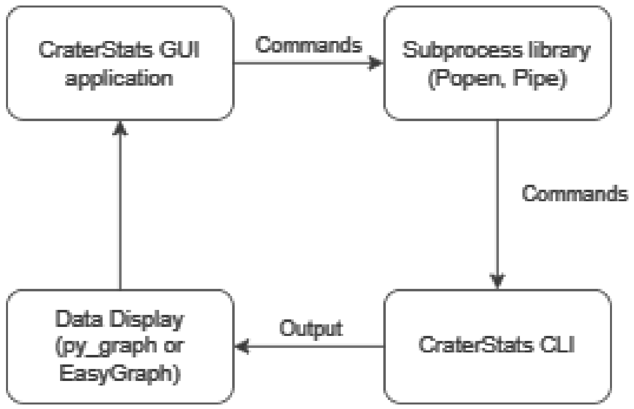
Table A presents each alternative we've considered with a ranking from 0 to 5 out of 5 for how well it performs for the desired characteristics we have. As seen from the data TKinter and DearPyGUI were obvious winners, with TKinter only winning in our choice because of our time to research and get familiar with the library.

Proving Feasibility

For this project to progress fluently and be successful, we must make sure that what we want with the program using TKinter is possible. To make sure of this we will be making a small program that exports graphs from equations that the user will input. This demonstration will give us the skills required to make sure that the craterstats program runs smoothly through the interface.

Technology Integration

Considering that we know of the challenges we face and how we plan on solving them, it is important to show how our solutions fit together to produce a GUI for the CraterStats command line application.


CraterStats GUI application
Commands
Subprocess library (Popen, Pipe)
Commands
Data Display (py_graph or EasyGraph)
Output
CraterStats CLI

The way our system works is by first taking commands and using Popen and Pipe from python's subprocess library. These commands are then passed to the CraterStats command line application. The output from the command line application is then fed into a graphing library (such as py_graph or EasyGraph), which will be used to display said data. After the data is displayed, the program continues to run. This process is similar to how the web application will run, as the web application will use the subprocess library to pass commands to the CraterStats command line program, and will use a graphing library to display data gathered from the program before continuing to run. This system will allow users to use the CraterStats application without knowing how to use a command line application, as it will provide a GUI for usage.

Conclusion

So, the problem we are currently trying to solve is to develop a graphical user interface for a command line-based crater statistics application. This problem is important to solve as, currently, the crater statistics command line application only displays statistical data. Creating a GUI for plotting that data makes it easier to understand, and makes the tool easier to use. With our application's interface, we plan on having a standalone window for it, a toolbar to provide access to functions and modify statistics, an area to display plots of relevant statistical data, and options for modifying data and functions.

Meeting our goal does present some technological challenges. For one, we need to find a way to connect both the command line application and our GUI application. One other challenge is running this GUI application on a web server. We must also find a way to best display our data. To solve these challenges, we want to use python's subprocess library to run the command line application and get its output to be displayed in a graph. We also want to use a server-less service, such as AWS Lambda Functions, which would allow us to run our application and maintain the same interface. Lastly, we want to use some python libraries (such as py_graph and EasyGraph) to plot statistical data gathered from the command line app.

For ease-of-use, we want to develop a user-friendly interface. Such an interface would also be easy to maintain, and would allow anyone to use the application. Our program must also have cross-platform support. To develop such an interface, we have several solutions: using Unity, Windows Forms, TKinter, DearPyGUI, ReactJS, or ThreeJS. One advantage of using Unity is that it allows for immediate UI creation, and relies on drag and drop style UI editing. One advantage of using Windows Forms is that it uses the .NET framework and is easier and faster to use. As for TKinter, it is available for use in many programming languages, and is also easy to use. TKinter also uses widgets for GUI creation. For DearPyGUI, it is useful for creating charts, tables, and drawings. An advantage of ReactJS is that it allows for dynamically creating UI elements. Finally, as for ThreeJS, it is useful for creating 3D designs and animations on a web page. After analyzing all of these tools, we came to the conclusion that the best to use are ReactJS, ThreeJS, DearPyGUI, and TKinter. We excluded Windows Forms, as it does not have the level of cross-platform support we want. In the end, we decided to use both TKinter and ReactJS, as the alternatives have too many problems working with the craterstats program.

Bringing this all together, we want to have our GUI application pass commands to the command-line application using python's subprocess library. After passing the commands and running the application, our program will get the output and use that to produce a graph. After displaying the graph, our program will continue running until it is closed.

Now, after all of the research our team has done, we have to move on to our next steps. Knowing what tools we want to use, we plan on starting the development of an interface. We also plan on further researching some of the tools we have chosen to use.

Works Cited

[1] B. Klein, "Tkinter - the Python interface for Tk | Tkinter | python-course.eu," 1 February 2022. [Online]. Available: Python-course.eu. [Accessed 21 March 2024].

[2] J. Hoffstadt, "Home · hoffstadt/DearPyGui Wiki · GitHub," [Online]. Available: https://github.com/hoffstadt/DearPyGui/wiki. [Accessed 21 March 2024].

[3] D. Takahashi, "Unity (game engine)," [Online]. Available: https://en.wikipedia.org/wiki/Unity_(game_engine). [Accessed 21 March 2024].

[4] "Windows Forms," [Online]. Available: https://en.wikipedia.org/wiki/Windows_Forms. [Accessed 21 March 2024].