Greengraph

Sofia Qvarfort
Student No. MPHYG001

January 10, 2016

Abstract

This is a complementary report for the `greengraph` software package submitted as coursework for a Python course in research programming. I outline some of the design choices made while packaging the software and outline the structure of the testing framework used. Finally, I discuss the benefits of preparing software for distribution and summarise the

1 Introduction

This is a short report written to accompany the small program `Greengraph` - a tool for counting the number of green pixels found in a series satellite images between two points.

2 Testing

There are several important aspects included in the testing.

The extreme cases are: when both start and end location are set to the same name. We have made sure that the program sets the step size to

I think I should be able to write both the tests and the visualisation in the same file.

2.1 Testing for bad input locations

This was a tricky one, since I did not find a good way of reporting on that error.

3 Design Choices

In this section, we shall outline some of the deisgn choices made for packaging of the `greengraph` code.

Let us begin with the partitioning of the classes. The code has been split into two classes, `Greengraph` and `Map`.

One difficulty found was choosing in which of these classes to implement new functions. The method `show_map` in the `Greengraph` class uses elements from the `Map` class, and manipulation of the Map type object, however, we made the choice to let any code interact primarily with the Greengraph objects, from which we can access methods in Map.

3.1 User Interface

greengraph can be accessed through a simple command line interface. I have strived for retaining as much user freedom as possible, so to allow for both fast and slightly more sophisticated use. re I chose to make the step size optional, to favour quick use when the user might just want results.

Furthermore, I have added the argument --delay which introduces a delay in how quickly the API receives requests (see more in the next section).

This version of the code takes three arguments: start, end and steps. I have chosen to let steps be optional to allow for flexibility. The default of the code

I chose to let the user input the filename and the extension. savefig() is a neat function that outputs an image based on the file extension. I personally prefer to be able to name the files myself, thus the choice.

I also chose to make showing the plot optional, since rapid work-flow can sometimes be interrupted by having to close unwanted pop-up windows.

3.2 Choice of licence

Following advice on GitHub, I chose to implement the simplest and most open licence as suggested by them, which is the MIT licence. The MIT licence is considered to be the most liberal licence out there. Since the aim a the moment is training and not much else, this was seen as a good choice. More information on the MIT licence can be found here: http://whatis.techtarget.com/definition/MIT-License-X11-license-or-MIT-X-license

3.3 Working with the Google API

As noticed during testing and building, the Google Maps API has a quota limit for the number of requests that can be sent to it during a short period of time. The API reacts by sending the image seen in Figure **??**, which restuls in a flat line at $y = 323$ pixels in the graph outputted by greengraph.

In order to catch the API overload error, two solutions were considered. Google recommends that a time delay is added to the loop, such that the API does not receive a high number of requests in a short period of time. Whereas it was not found to make a substantial difference once the delay was invoked

Instead, we have chosen to implement a simple function that checks when some of the values are equal to 323, which is the number of green pixels in **??**. We chose the threshold value of two and to print a warning instead of quitting the program. The value error_threshold =2 was chosen because it would be extremely unlikely that two map requests contained a number of 323 green pixels, unless the API limit was hit. This way, the user will know that data in the graph does not contain real data, but rather data from the error picture supplied by the API.

I chose to reduce the test of the APi overload function to checking an aray that contained the value 323 more than once. The ideal case would have been to feed the API eror image to the program, but due to time constraints this could not be done.

The `time.sleep()` does not currently raise a value error for negative input. Therefore,

4 Discussion

In this section, we discuss the advantages and disadvantages of the work related to packaging software into a presentable format.

Advantages The foremost advantage of packaging the code into ready use is that any mistake or irregularity present in th ecode is more likely to be caught as the developer thinks about the usages of the software.

Any automated tests that are written can easily be run at a later point when the software has been expanded.

Using the framework of licenses and README files also greatly helps others who potentially wish to use the software.

Furthermore, new users of the software will have an easier time familiarising hemselves with the software if the code is structured and simple. Further uses of software such as SPHINX will make the process easier. By being ‘user-friendly’ the software is also more likely to spread and be used more often.

Disadvantages In my opinion, the disadvantage is the time required to properly package the code. In the context of research programming.

5 Community building

The existence of a user base is invaluable to a developer or research programmer

Each scientist specialises in one particular area, which means that any code they write will be highly specialised. I beleive that a challenge for researchers is to reach beyond their surroundings and adapt their potential software to a wider circle of users.

I believe that it is important for scientists to think about how their software can be used in other instances. The cross-pollination of scientific fields can sometimes lead to great advances. To take an example from my own field; the advances in quantum computing using Nitrogen-vacancy centres have recently been shown to ofer great advantages for quantum sensors. Thus, keeping n open mind regarding neighbouring fields and devoting some time to ponder on how one’s software can be of use to others is of great importance. In the best of scenarios, perhaps only little modification is needed.

This is where building a community of users can become invaluable, as their own specialities and research might come to influence and add to the existing software.

In a scientific context, one step towards building a community of users is to purpose the code for wider usage. Naturally, an element of networking at conferences and across universities can also help.

6 Conclusion

In this report, we have outlined the steps in packaging