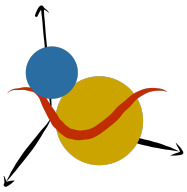
Software Testing Plan

March 26, 2021

Team Triaxis



Sponsors:
Dr. Will Grundy, Dr. Audrey Thirouin

Faculty Mentor:
Sambashiva Kethireddy

Team Members:
Eleanor Carlos
Reyna Orendain
Andres Sepulveda
Brandon Visoky

Overview

This document outlines activities that are aimed at ensuring that our project's implementation exhibits the necessary functional and non-functional characteristics. In it, we describe how we intend to ensure that the expectations presented in the requirements and design specification documents are met, via a well-planned software testing regime. Tests are divided into three categories: unit testing, integration testing, and usability testing, all of which encompass the various ways we will verify functionality for our development this semester.

Table of Contents

Introduction **3**
Unit Testing **4**
Ellipsoid 4
Fluid Equilibrium 4
JSON Parser 5
Integration Testing 7
Ellipsoid 7
Fluid Equilibrium 7
JSON Parser 8
Usability Testing **9**
Ellipsoid & Fluid Equilibrium 9
JSON Parser 9
Conclusion **11**

Introduction

Exploring our galaxy is a journey that often feels like one that is beyond any of our lifetimes. Something that can only be achieved through science fiction, television shows, and novels. While this may be true at present, there is amazing work being done every day by scientists and researchers to observe the galaxy and turn what often feels like fiction, into reality. One of our only ways to study and research celestial bodies is to visualize them based on observed data. This is especially true for objects that are far too small and distant from Earth to even consider sending a man-made craft to study them. Of particular interest to our project, researching asteroids relies nearly entirely on different forms of visualization because of their distance from our planet.

We are Team Triaxis and we are working with our clients, Dr. Will Grundy and Dr. Audrey Thirouin, on the project “Complex Asteroid Shapes in Modeling of Binary Asteroid Systems.” We are responsible for delivering the goals put forth by our clients and Lowell Observatory. Of these goals, Dr. Will Grundy and Dr. Audrey Thirouin are focused on modeling binary systems in the Kuiper Belt, and the Licht-cpp software that we are continuing work on helps them visualize and model the data from light curves in ways that would otherwise be impossible.

This is the third year and third iteration of this project given to us by Lowell Observatory. Team Triaxis looks forward to continuing the development of this project as left to us by our former peers, Paired Planet Technologies and Team Andromeda.

Our iteration of this project includes implementing a new module to render binary systems observed by Lowell Observatory. Our primary objective is to render the components of these systems as triaxial ellipsoids, a more complex model than the current spheres they are rendering them as now. While our primary objective is to implement the ellipsoid module, we will also be expanding upon previous work to create a module for fluid equilibrium, as well as a graphical user interface (GUI) to vastly improve the functionality of licht-cpp as it was given to us.

Part of delivering the highest quality product we are capable of delivering for our clients is to develop and expand on the already robust testing suite that has already been developed for Licht-CPP. The framework for unit tests has already been developed by previous semesters, and thus our largest goal is to incorporate the new features and fixes we have developed into the existing suite of tests that already exist.

Unit Testing

Unit tests are invaluable tests as they are able to test each individual function our team has developed throughout the project. They are simple and quick to run, and allow for easy validation depending on how complex an individual function is.

The functions our team will be creating unit tests for are the ellipsoid module, fluid equilibrium module, and our JSON parser.

Ellipsoid

The Ellipsoid module was originally designed and implemented by Team Andromeda, and thus already contains unit tests for its functionality. We will improve on these tests where possible, but in summary, these tests include:

1. TestShape Test
    a. **Description:** Create and validate ellipsoids
    b. **Main Flow:**
        i. Generate Ellipsoid Object
        ii. Validate the center and radii are correct.
        iii. Change radii values
        iv. Validate center and radii are still correct.
    c. **Expected Outcome:** Valid ellipsoid object created.
2. RayTracing Test
    a. **Description:** Ensure ellipsoids return a valid light curve
    b. **Main Flow:**
        i. Generate ellipsoids.
        ii. Initialize values for tracer object.
        iii. Initialize the tracer with the ellipsoids.
        iv. Validate the tracer receives a lightcurve.
    c. **Expected Outcome:** Ellipsoid is successfully traced in the ray tracer.

Fluid Equilibrium

The Fluid Equilibrium module essentially uses the same code as the Ellipsoid module, and thus most of its tests are the same and need only be checked for validation that it is working correctly. These are the same aside from one extra unit test in the ChiSquared module.

1. TestShape Test
    a. **Description:** Create and validate objects in Fluid Equilibrium.
    b. **Main Flow:**

i. Generate Fluid Equilibrium (Ellipsoid) Object
ii. Validate the center and radii are correct.
iii. Change radii values
iv. Validate center and radii are still correct.

c. **Expected Outcome:** Valid ellipsoid object created.

2. RayTracing Test
    a. **Description:** Ensure objects in Fluid Equilibrium return a valid light curve
    b. **Main Flow:**
        i. Generate objects in Fluid Equilibrium.
        ii. Initialize values for tracer object.
        iii. Initialize the tracer with the objects in Fluid Equilibrium.
        iv. Validate the tracer receives a lightcurve.
    c. **Expected Outcome:** Ellipsoid is successfully traced in the ray tracer.
3. PenaltyFunction Test
    a. **Description:** Ensure the amoeba function is penalized for incorrectly calculated values
    b. **Main Flow:**
        i. Generate object in Fluid Equilibrium
        ii. Call function for checking fluid equilibrium
        iii. Validate that the fluid equilibrium penalty value is correct based on radii, strength of fluid equilibrium, spin frequency, and density.
    c. **Expected Outcome:** Correct penalty for chi-square is calculated with values input.

JSON Parser

The JSON Parser is what we have developed to replace the old UserInput file used with the NLM. While tests previously existed for the UserInput file, because the JSON Parser ingests data differently, we cannot use the previously developed tests.

1. verifyJSONDataTest
    a. **Description:** Verify that the JSON Parser correctly identifies a missing file. The test will be trying to pull a “DoNotRead.json” file that does not exist in the directory.
    b. Main Flow:
        i. Run the readJSONUserInput function with the “DoNotRead.json” filepath passed in.
    c. **Expected Outcome:** Error message reporting the user entered an incorrect filepath, or that there was an error in trying to find the JSON file.

2. getJSONDataTest

a. **Description:** Verify that the JSON Parser correctly parses through data. A purposely messed up UserInput.json file will be provided in the test directory to verify error handling works as intended.

b. Main Flow:

i. Open UserInput.json file using nlohmann::json parsing module

ii. One or more variables will not be of the proper type. These can be any of four types: Integer, String, Double, or Boolean

c. **Expected Outcome:** Error message reporting the user entered an incorrect type of value into a parameter that expects another type (i.e. bool vs. double).

Integration Testing

Integration testing is used to ensure that the interaction between different modules is working as intended. While unit testing is valuable to test each individual function, integration testing tells us whether or not functions provide the correct functionality once working together.

Due to the way that the Forward Model works, it interacts with most of the different modules within its single function, allowing for very thorough integration testing through the single testing file. Of course, this is a half measure, and we will be ensuring that the existing integration tests work as well as building new ones to test the functionality of functions that do not go through the Forward Model.

Ellipsoid

The forward model is a perfect place to test the integration of the Ellipsoid module, as it is the place that provides the data to create and interact with the ellipsoid shape. It does so by receiving the data used to create the ellipsoid, creates the ellipsoid, and then passes it off to the tracer function to be rendered. The tests for these ellipsoids already exist inside of the testing suite, but we will expand upon them where possible.

1. Test Forward Model
    a. **Description:** Ensure that the Ellipsoid receives a valid light curve when used with forward model
    b. **Main Flow:**
        i. Input the parameters required for the forward model to operate.
            1. These tests are real-world examples of asteroids, which serve as extra validation that the modules are functional.
        ii. The forward model creates the ellipsoid and provides the resulting light curves of the ellipsoid shapes.
        iii. Validate the light curve exists.
    c. **Expected Outcome:** Light curve data is produced after creating Ellipsoid.

Fluid Equilibrium

Due to the Fluid Equilibrium module being essentially the Ellipsoid module, we can use most of the same tests for integration for both, with the exception of testing the ChiSquared penalty function.

1. Test Forward Model
    a. **Description:** Ensure that the Fluid Equilibrium object (Ellipsoid) receives a valid light curve when used with forward model
    b. **Main Flow:**

i. Input the parameters required for the forward model to operate.
    1. These tests are real-world examples of asteroids, which serve as extra validation that the modules are functional.
ii. The forward model creates the ellipsoid and provides the resulting light curves of the ellipsoid shapes.
iii. Validate the light curve exists.

c. **Expected Outcome:** Light curve data is produced after creating an object in fluid equilibrium as well as the correct penalty value for not fitting in fluid equilibrium.

JSON Parser

To make sure that the JSON Parser is taking in the right type of value in a parameter, the JSON parser will try and catch any value that isn't acceptable. When running the NLM, each variable checks if it's a match to the ones in the Forward Model. This is to ensure that the values are passed in correctly to each corresponding variable.

1. Test Non-Linear Minimizer (NLM) -
    a. **Description:** Ensure that the input in the JSON Parser matches up with what the NLM is expecting to be able to do the calculations for either the faceted objects, spheres or Ellipsoids (Triaxial and Jacobi)
        i. **NOTE:** This test will most likely be deprecated once it's fully proven that it passes with the current integration. As soon as it is confirmed that the UserInput.json works as intended, the UserInput.txt will be considered obsolete and no further comparison with the .json file is required.
    b. Main Flow:
        i. Run readJSONUserInput function to successfully pull data from UserInput.json.
            1. Data will be stored in a similar structure to FMArguments (used for Forward Model / NLM calls) to easily compare data.
        ii. Run readUserInput function to successfully pull data from UserInput.txt. This data will be stored in the FMArguments struct during testing.
        iii. Compare values iteratively and ensure all 63 current parameters are matching up correctly for use later in the Forward Model / NLM
    c. **Expected Outcome:** The values pulled from the UserInput.json file exactly match up with the values expected from an input using UserInput.txt. The UserInput.txt will then be obsolete.

Usability Testing

Usability testing is what allows us to ensure that our software is easily navigated and usable by users. Because of this, we are unable to directly program these tests and instead must come up with plans to gain approval from our clients once each module is completed. To fail these tests means that our software is unusable from a user's perspective, and we would like to ensure that our product is as user-friendly as possible.

The users interact with our codebase through a command-line interface as well as a User Input file via the NLM. Since these are the only two points of access for our users, it makes usability testing much simpler and allows us to really focus on how to improve/test them.

Ellipsoid & Fluid Equilibrium

The Ellipsoid and Fluid Equilibrium modules are simple extensions of the forward model and serve as a way to extend the functionality of the program as a whole with new shapes. These are not expected nor intended to see any user interaction aside from the user passing in data to create the shapes themselves.

Thus, we must ensure that the ways our users can create these shapes are easily implementable and obvious to interact with. Thus, our usability tests for these modules instead come from the JSON Parser, which is described next.

JSON Parser

The reason the JSON parser was implemented was due to the fact that the previous user input file used by our clients was itself not very user-friendly. This is one of the few ways our clients are able to interact with the system itself, and therefore it is very important that they are able to use it without any difficulties.

We are fortunate to have developed this new JSON user input file directly with our clients and have developed it with their feedback along the way. Therefore it was designed with their ideal use cases in mind, and so hopefully is highly usable and familiar to them.

Our usability tests for this module then are best designed to have our clients work directly with the new user input file themselves. We then can get a judge on how easy it is to navigate through and use, as well as determine exactly how much more usable our new implementation is to use vs. the old solution.

Once we observe our clients using the new file, we can ask basic questions such as:

- How easy was it to interact with the JSON file?

- Did you have any difficulties?
- Is there any functionality you find missing from this solution?

With these questions answered, we can then adjust and modify our implementation to further match what our clients are expecting and would approve of. Because of how closely we are working with our clients, this has been an iterative process, and the usability testing has been an ongoing process all semester.

Conclusion

Our knowledge and understanding of space is continuing to grow with the technology we develop to observe it. Thousands of years ago, humanity did not have a fraction of the understanding we have of outer space now. Imagine what we will know in another thousand, hundred, or even several years from now.

This growing knowledge pivots and relies on the thousands of astronomers, including our clients Dr. Will Grundy and Dr. Audrey Thirouin's, continued research. Of course, this research also depends on engineers and programmers who can design and build the specialized tools that are needed day to day.

Asteroids can be a very important and interesting field of study for astronomers for a multitude of reasons. Since they are much smaller than planets and far too numerous to land spacecraft on each one, researchers must rely on technology like licht-cpp to continue their research. This is why it is also so important that we are able to fully test and validate this product - so that we can ensure our clients are getting accurate and correct results during their research.

The unit, integration, and usability tests as described above have been implemented and designed with great care and attention to help deliver the best product possible. With our unit testing, we are using the same tests developed by past teams to ensure that data is processed and interpreted accurately. In our integration testing, we are making sure that additions to the program work properly with the forward model. Finally, with our usability testing, we want to make sure that the new user input JSON file is up to the standards of our clients.

We hope that this product will not only meet, but exceed the expectations of our clients, and these tests are but only a single part of that goal. We look forward to implementing these tests and working further with our clients to improve them in the future.