**Code Review Grading Rubric**

Total Points: / 500 points

Week: 8 Nov - 15 Nov

Team Name: Raytheon

Preternship Software Engineers: Justin Pajak, Patrick Creaven, Carter Goldman

Preternship Project Manager: Ryan Farrow

Proposed Change List:

* Get satellite data file and figure out how to parse and implement it in our project
* Write code for the conversion from spherical orbit data to xyz relative to the center of the earth
* Utilize a clock library in our program in order the display a satellite’s changing location over time, change the position of the satellites in the graph

Deliverables (250 / 250 points) :

* Not accomplished--unable to install libraries on student machine due to lack of privileges. Working around by hardcoding values
* Not accomplished--same issue as before. Using floating point XYZ variables to account for this
* Accomplished--love it! But do be careful with it when debugging. If your code behaves unpredictably, this is a likely culprit.

Design ( 150 / 150 points) :

* Code Clarity-
  + Nice work with the #define for your constant. That’s generally good style for C++.
  + Would like to see at least minimal comments in some of your classes. I get that it would be a waste of time to label every single getter and setter, but even just a comment on top of your first getter that says // \*\*\*Getters\*\*\* or something similar would help structure things in my opinion.
  + Love the descriptive variable/function names! Obviously, you don’t want them to be too long, but I think you’ve struck a good balance.
* Code Intention -
  + Glad to see the set\_xyz function included in there! I think that’ll make your code more readable and save you some trouble with debugging.
* Code Integration -
  + I understand you’re having trouble with the Dijkstra’s algorithm implementation. That’s pretty frustrating, but let’s keep working on it and come up with a worst-case scenario plan in case you get caught by the deadline and have to whip up an inferior end product. I’m thinking something like a naive closest-first algorithm or something. We can discuss--just don’t get too caught up on this and make sure you have a product to deliver by the end of it!
  + Aside from that, it seems like everything is fitting together in your main file pretty well, glad to see it!

Spiral Software Development (100 / 100 points):

* Objectives -
  + Make our implementation of Dijkstra's Algorithm work within our use-case
  + Develop code that allows the user to designate the location of a new ground station and choose what groundstations they would like to the know the latency of communication of
  + Figure out how to pause the satellite data updates with a keyboard press in order to take user input
  + Make the code look pretty by fixing spacing, indentations, and adding comments
* Risks and Alternatives -
  + The Dijkstra's Algorithm implementation we have may prove to be unusable or difficult to use in our use case
    - Alternative: Build a new Dijkstra's Algorithm implementation from scratch
    - Alternative: Implement some other shortest path algorithm
    - Suboptimal Alternative: Hard-Code values to at least get a somewhat working finished product
  + Threading on C++ may be somewhat funky, specifically concerning the user interaction with the program
    - Alternative: Re-design the user interface to have it increment at a set time interval, and continuously ask the user at each interval if they wish to continue, or to calculate the desired latency
* Plan for Product Development -
  + Use GDB along with other debugging methods to pinpoint the exact problem with the Djikstra’s Algorithm implementation as it is written; and/or rewrite it from scratch
  + Utilize C++ thread library to allow for intended user interaction
  + Use a refractor to clean up the tabs and spacing in all code files. Add more comments where they are deemed necessary

<http://www.cplusplus.com/reference/thread/thread/>