**Code Review Grading Rubric**

Total Points: 500 / 500 points

Week: 2 Nov - 8 Nov

Team Name: Raytheon

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

Preternship Project Manager: Ryan Farrow

Proposed Change List:

* Utilize inheritance to make the satellite and ground station classes have a common parent, making it easier to facilitate calculations.
* Create functions in satellite class and ground station class that allow basic communication between one satellite and one ground station.
* Create an initial graph data structure to house the satellites and ground stations, so that we can represent their relationships to each other.

Deliverables (250 / 250 points) :

* (Note to self: discuss C++ portability)
* Make satellite/ground station classes: complete
* Create functions in satellite class, ground station class: complete
* Create initial graph data structure in project.cpp: complete
* Also did some extra work from next week’s goals
* Ran into some issues trying to integrate a third-party library--compiler issues are the culprit. Looking for a workaround now.

Design ( 150 / 150 points) :

* Code Clarity-
  + “int c = 30000000” is not great. It’s a global variable in project.cpp, so you need to be mindful of two norms for global variables: first, they should have all-caps names, and second, those names should be a tad descriptive with underscores for spaces. As an example, int PACKET\_SIZE = 1024; is a good declaration of a global variable because it’s descriptive and you know it’s global. Side note: the reason you make it all caps is because there is a norm around all-caps variables not changing. You NEVER want a global variable to change; if your code has that, you need to restructure it or you’re likely going to have a huge mess on your hands. Sorry for the essay, but this is the sort of thing where good style saves you headaches down the line.
  + Make helper functions to consolidate some of this code. You have the “set\_satellite” function being called three times per satellite, which could easily be condensed into one “set\_satellite\_xyz” or similar function.
  + More comments, please!
* Code Intention -
  + Intention of your code is fine; just keep in mind that if you don’t follow good stylistic procedure, you’ll start seeing issues with your code intention, especially as it scales up. For now, looks good!
* Code Integration -
  + The project is scaling up quickly, but it looks like you’ve structured your classes thoughtfully so that the moving pieces will come together nicely in project.cpp. Good job! Let me know if you have any issues with this; definitely keep in mind the types of your function inputs and outputs as you continue.

Spiral Software Development (100 / 100 points):

* Objectives -
  + Make the Graph\_Sat class more concise by eliminating redundant functions
  + Fix the Dijkstra’s algorithm function in the Graph\_Sat class so it doesn’t segfault when there are more than 2 satellites in the constellation
  + Utilize constellation specific TLE data, as opposed to the full list of current satellites (can perform more realistic communications between cubeSats)
  + Change the Project.cpp file so it now uses the Linux epoch time rather than usleep()
  + Work on finding and integrating a third-party library that works with the compiler we have to perform orbit calculations; if not, perform calculations on our own
* Risks and Alternatives -
  + We may not be able to figure out how to fix Professor Morrison’s Dijkstra function so it doesn’t segfault
    - Alternative: we may have to create our own from scratch to tailor it with our specialized Graph\_Sat class.
  + If we wrote our own library for orbit calculations, it may be less accurate or robust
    - Alternative: attempt to modify an existing library to be compatible with our existing program, if we can’t find one that will work with our C++ installation
* Plan for Product Development -
  + Add a set\_xyz function that calls the individual set x, y, and z functions
  + Investigate the existing Djikstra’s algorithm implementation to figure out where the segfault is
  + Simple switch from all current satellites data file to constellation specific one
  + Simple switch to performing calculations based on Linux epoch time
  + Keep researching to find libraries; there are more robust libraries written in other languages that we may be able to implement, or we may just have to modify the one we found