Mars Rover Simulation Discussion

This Mars Rover simulation project was built for the purpose of learning to communicate using network sockets in C++. Ben and I were ultimately able to build a server and client which are able to form a connection and exchange data, although we encountered difficulty in some situations. For instance, when we attempted to send the .JPG file to the client, the output was always corrupt even though the file was of the correct size. By the time we finished the project, we felt frustrated with our language of choice, feeling that this task would have been much easier to accomplish in Java. We went into the project expecting that a C++ implementation would be somewhat more difficult, but we underestimated the problems that would arise. One of the problematic issues is how to represent the data when sending and receiving it. C++ has no built-in byte type, and so we decided to use an array of chars to represent our data. This led to us using several operations from the string library, which tend to do things like append a \0 byte to the end of the character array. In the end these small additions were enough to corrupt a JPEG image which is obviously more sensitive to corruption than plain-text is. Some things we attempted while debugging the image transmittal were having the server write out the image before sending it to the client (successful), and having the server send a text file to the client (successful). While I think it is very likely that Ben and I could figure out our errors given more time, we found ourselves in a bit of a time crunch with final exams and the last-minute power outage.

Other considerations we had to take during the design and development of our project were: how to split the work, how to format our headers, and how to ensure that the read() and write() calls were looping in order to consume their entire buffer. The most obvious way to split the work was to have one person do the client and the other the server, and although this worked for the first 70% of the project or so, more collaboration was required to bring everything together and troubleshoot issues with the image transmittal. In a way this is a good thing because it forced us to become very familiar with each other's code, and as a result we both have a very good understanding of our project. We coordinated our activity using a combination of github and facebook chat, and since we were often changing the same code since the project is so small we had to take turns making commits. Often when solving bugs, we would post the current best working code on github and then both tool around with it while chatting so that we could follow up on our own ideas while sharing information. Ben decided the format for the messages including the headers, which is to use a single-character ID field, a length field, and a delimiter token (!). In retrospect I think it may have been easier to use a fixed-length header with a predefined length field, and this may have prevented some issues with handling messages over the network. After defining this message interface, it was a fairly simple task to have the client ask for input and send that over to the server, which responds according to the relevant switch case. In some cases (when sending an image), the messages are longer than our client's read() buffer size, and so multiple reads must be made. Each time the client loops it fills its buffer with data from the socket stream, and then writes that data to an accumulator called “msg” that eventually holds the complete message from the server. A similar problem occurs when writing data from the server, and so we made sure to calculate the length of an outgoing message and pass that value to the write call until all bytes were written. We found the examples posted online to be extremely helpful in the establishment of our network sockets, and you can see the structure of those examples reflected in our project because we made no attempt to recreate the wheel. In the end we were able to successfully use our client and server over the network, and we are both glad to have this code for future use and improvement.