**iOS Final Reflection**

I was inspired to take up writing a telegram translator application after seeing an episode of the TV show *Sherlock,* in which Watson sees Sherlock flashing a light in certain intervals that seem to represent Morse Code. After seeing this, I wanted to create an app where one could type in any english text, encode to morse, and have the code flash as sequences of light, beep in a series of sounds, or just be printed out as string of dashes and dots. This app is compatible for two-way translation (morse to english as well as english to morse).

In APCS class, we had written a program which took English text and translated it to Morse Code. Getting this basic translation to work was just a matter of translating what I had previously written in Java into Objective C. The underlying data structures are arrays and treemaps. I had an array coded as a binary tree so that the placement of a character in an index was based on the relative frequency of occurrence of the letter in the English language in an attempt to speed up performance of translation. However, after doing that, I decided to go a few steps further by increasing the compatible characters available for translation and implementing multithreading to improve efficiency. Learning basic multithreading in Objective-C proved to facile. The main purpose was to perform tasks in background threads so as not to slow down the performance of the tasks at hand in the main thread.

I then designed the User interface with two text fields (one to enter for english text on top and one to view morse text on the bottom), a translate button (which takes whatever is in the top text field and translates it for the bottom text field), a swap button (which cause the bottom text field to replace the top text field and vice-versa), a segmented button (which allows the user to choose how he wants the text to be translated [either as text, sound, or light]), and finally a slider (to adjust intensity of the light or the volume of the sound).

Implementing the light and sound functionality was a matter of using AVCaptureDevice and AVAudioplayer respectively. The difficult part was getting the timings all correct which I solved by using the “performselector afterdelay” command.

In addition, I had multiple views stored in a navigation controller. I had a home screen which has an image of an example translation and two buttons, one that allows you to access the translator and another that takes you to a UITableView which shows a Morse Code reference table. The color scheme of the app was chosen to be blue to mirror Google Translate in some ways.

In the end, I am satisfied with my final project and look forward to work on future iOS apps.