This past summer I interned for Dr. Stephen Carl in the Computer Science department at Sewanee from August 10th through August 21st 2015. We started out with a two-dimensional array of cells that, based on a given set of rules, can generate a grayscale image that is constantly changing. This was a project that Professor Carl had a student build a few summers ago, but now he wanted to make it more aesthetically pleasing and add an audio element. The language we worked in was Java in a Processing environment and Dr. Carl had already written a prototype code for adding audio that operated in a language called Pure Data. His prototype worked by sending Pure Data the information for the 2D array of cells. This was bulky and I wanted to make it run entirely within Processing so that it could be exported as Javascript. My last responsibility was to improve the code itself by organizing it into tabs, adding comments, and removing extraneous bits. These three goals of illuminating the image with color, sonifying the cells, and organizing the code were what I set out to do with this project.

Since Professor Carl and I discussed this generative art project in early May whereas the internship was not to take place until mid-August, I had most of the summer to familiarize myself with the concept and form a general idea of how I wanted to go about building the final product. Dr. Carl also gave me his original version of Waves.pde—the grayscale piece—which would become the template for my project. In order to improve the aesthetic I decided to add color to the previously grayscale piece and have the cells change shape as the color rippled through them. The color would also provide a medium for me to add sound by linking the frequency of the noise values to the red, green, and blue values of each pixel. The majority of the
summer before my internship officially began was spent considering ways to accomplish these goals by examining Dr. Carl’s template while simultaneously refreshing myself in Processing.

On August 10\textsuperscript{th} I began by heavily picking at the code by commenting out lines to see exactly how it worked. Since the final product would require a lot of computing power, I wanted to remove as much extraneous code and variables as possible. I also researched other classes that I could use to replace our homemade cells class altogether and, in particular, particle systems. In other scripts that used this class, I noticed the particles included a pre-programmed general set of physics which could work to improve the rippling effect. After reading the Processing Javadocs on particle systems, I discovered that I needed to rewrite the cells class to be a subclass to particle systems in order to retain the physics elements of particles because the particle systems class did not have the necessary methods. I reviewed my understanding of subclasses and super-classes before beginning to reconstruct the sketch. However, once they were implemented, each particle’s agenda demanded too much computing power and the image would stutter—despite still being in grayscale.

While researching different shapes in other Processing scripts from the Processing handbook by Casey Reas and Ben Fry and looking at examples from Matt Pearson’s book Generative Art, I found a class builds polygons with a constructor for the number of sides the polygon would have. I thought this would be promising since Professor Carl wanted the cells to change shape depending on the ripple. I manipulated the variables in their polygon class and implemented it as the constructor for each cell. The RGB value would dictate the number of sides, in such a way that an intensity of 255 would yield almost a circular polygon and a black
cell would be a triangle. After they were implemented into the cells class, I could implement color to each cell. Essentially I am creating three different layers of rippling—red, green, and blue—to constitute the RGB values of a pixel. This required tripling most elements of the pre-existing code and tweaking the empirical formula for the ripple that we were using. Unfortunately, reshaping the cells with color was too processor heavy, so I scrapped the reshaping and stuck to an offset lattice of hexagonal shapes.

Once I had implemented color and given the cells a crystalline pattern, it was time to begin sonification. I researched different Processing libraries used for sound and, while ESS was the one we used in CS276, it is not compatible with Processing 3.0. Processing comes with a sound library itself, but there are not Javadocs to accompany it. I chose Minim because it has regularly updated documentation and it came with the methods I would need. To link sound with the RGB values, I used red for an upper-range frequency, green for a mid-range frequency, and blue for a mid-bass frequency. This way the sound would come out as fluctuating as the ripples of color moved across the canvas. However, since three new oscillators were constructed for each cell without being stopped, the noise sounded like a screeching cricket. After manipulating them so the volume would be controlled during each iteration of the script, the number of computations the computer performed each second was too much and this sounded choppy. I settled on three oscillators in the end that worked by averaging the intensity of each color throughout the cells.

This research experience taught me a great deal about calculating and managing the limitations of working with computers in a way that I will be able to incorporate into all my
future projects as a computer science major. In terms of career goals, this research opportunity showed me a creative side to the computer science field that I thoroughly enjoyed. Since completing Professor Carl’s project, I have begun working on two other generative art pieces in my own time so that I may have a portfolio of them to present in a professional work environment.