Jay Annadurai

MAD SP23

T. Faas

16 Jan 2023

Personal Composition: “Proto-DNA”

1. Link to Github: <https://github.com/annaduraij/N220LabsS23>
2. Process:
   1. My Personal Composition, titled “Proto-DNA” is inspired heavily by my background in biology but also the repeating-pattern nature that is a motif of p5.js art. The double stranded helix is one of the most organized and repetitive structures I’m familiar with and as such I began pursuing a design that was at least evocative of it. My initial plan was to have staggered, static sine waves printed horizontally across the canvas but as fate would have it, when I searched ‘p5.js sine wave’, the first link (<https://p5js.org/examples/math-sine-wave.html>) pointed me towards Daniel Shiffman’s animated sine wave and I was enthralled with it.
   2. The code seemed simple enough so I began reverse engineering the program to better understand it. I immediately discovered the comments he included, while helpful, were not comprehensive. Nevertheless, the program is not overly complicated but I clearly did not appreciate the elegance of the solution at first glance. Daniel uses an explicit for loop to compute and plot a sine wave but what perplexed me is why angular velocity was defined and implemented twice—once formulaically as dx = 2pi/T and again directly as an iterating theta value. Playing with the values in the code I understood the draw() function was being used as a second outer loop to animate the inner for loop; the theta was used as an angular velocity of the animation itself and the dx corresponded to the angular velocity and frequency of the wave itself.
   3. Once I understood the general algorithm that Daniel used to generate the animated sine wave, I rebuilt it with nomenclature more familiar to me. After debugging some issues where certain segments of code did not function outside of the setup() function, I was able to get the program online. After this point, it became much easier to experiment with and I was able to start adding several features:
      1. Cosine wave on top of the innate sine wave
      2. Multiple wave functions with phase shifts
      3. Animate the shape of the wave particles proportionate to its position in the sine wave
      4. Create two opposite waves and have line segments running between them to get to the DNA shape
      5. ‘If’ Control Structure to Change Colors
      6. Mouse-click to alter program parameters, specifically color ramping and wave animation speed
   4. Essentially the code uses a buildWave function to mathematically compute the locations and coordinates of points with loops and then stores them in specific arrays. While a multidimensional array could have been used here, I was not confident enough at this point to do so and rather implemented several arrays of the same size to store information. Next, the renderWave function contains the drawing aspects of the program and draws both the shifting particles and the lines from waveA to waveB.
3. Reflection:
   1. ‘draw()’ is an innate loop that can be used to the advantage of the programmer
   2. Sequential order of elements is critical—if one wants to refresh the frame, they must place the background not in the ‘setup()’ but rather the ‘draw()’.
   3. There is a computational limit but there are always alternative ways to increase modularity of the code, i.e. a cosine wave can be represented by a phase shifted sine wave
   4. There’s a lot of room for modularity and I can definitely improve the modularity and efficiency of the code by limiting redundant groups of actions
4. Planning Material
   1. I used the p5.js online sample code editor on Daniel’s sine wave function page to play with the code to understand it further
   2. For this project, I largely just played with the code to implement my own features until I got an idea of what I liked and what I didn’t like