N220 Midterm

https://dmrichey.github.io/N220Spring2021/midterm/index.html

Considering the six criteria for the application, I was able to split them into two halves. The first of these involved drawing the circles to the screen with the correct size and shape considering their relation to the position of the mouse. The second was the functionality that occurred when the mouse button was pressed down, in which the circles would animate and move down the screen.

To accomplish the first half of this, I first created three arrays. The first contained circle objects with properties for their X coordinate, their Y coordinate, their color, and their size. The Y coordinate, color, and size were all initialized to zero as a placeholder, to be filled as the program ran, but the X coordinate was declared to have the ten circles evenly split across the canvas. The second was an array of ten possible sizes the circles could be, and the third a similar array but containing strings for the possible colors of the circles. After setting up the arrays and the canvas, the program records the position of the mouse and determines how many circles are to the left of the mouse by iterating through the circle object array and comparing the x coordinate of each circle to the x position of the mouse. This step also determines whether or not the mouse is directly on top of a given circle to use as a center point for the changing sizes and colors. After the splitting point of the array is determined, the circles are drawn to the screen. The y position of the mouse is first recorded to the circle object. Two for loops are then used to iterate first from the position of the mouse to the left of the screen, and then from the position of the mouse to the right of the screen. This allowed for the mirroring of sizes and colors on either side of the mouse. If a center circle is discovered, then it would be drawn first and the two for loops would then iterate as normal.

To accomplish the second half, I attempted to use a while loop with the mouseIsPressed p5 condition. This while loop would run for as long as the mouse was held, moving the circles down the screen by five pixels each iteration until the circle reached the bottom of the canvas. Upon releasing the mouse button, the while loop would have ended, allowing for another iteration of the draw function, bringing the circles back to the position of the mouse. However, an issue was discovered with this method, namely that pressing the mouse button would cause the webpage to crash. It is unclear to me the exact reason this causes the webpage to crash, though I would assume that it has something to do with the loop not having a clear exit. I believe that I would need to do more research into how JS deals with while loops to have an understanding of what needs to be changed to make this work.

I referenced the p5.js documentation for clarification on the syntax and usage of some functions and conditions, such as mouseIsPressed, to verify that I was using them correctly and efficiently.