Jay Annadurai

MAD SP23

T. Faas

6 Feb 2023

Exercise 3: Objects

[Link to GitHub](https://jannadurai.com/Exercises/Exercise%203/index.html)

**Introduction**

The Object Exercises includes two exercises that retrofit objects into previous exercises and a new exercise to draw a set of user-interactive objects. While the exercises themselves are not fundamentally different, especially Ex3.1 which retrofits objects to the already-modular function-based Ex2.1, there is vast functionality attained with an object-oriented programming approach. The functionality is attained with the ability of objects to both encapsulate data as attributes and functions as methods which allows for incredible modularity. While Ex3.1 retrofits the rocket function to a rocket object with a draw method, and Ex3.2 introduces the Rectangle Class, it’s Ex3.3 that is the highlight of this set of exercises! It introduces a Vector class and a Color Fade class on top of the Circle class.

**Algorithms & Planning**

All files do contain a similar structure of declaring variables outside of the setup() p5.js function, initializing them with values and potentially objects within p5.js's setup(), and then drawing pertinent and dynamic elements within p5.js's draw() function. Functions and Classes are written in a separate section after the p5.js draw() section. All Classes contain similar drawing properties which are added via the constructor and then stored within the object.

**Reflection**

With Ex3.1, the retrofit of the function-oriented Drawing Drawer into the object-oriented Drawing Drawer, there was not much initial difficulty. Any temporary variables that stored individual properties of the ‘parts’ of the Rocket were shifted into attributes. And this area contains the greatest potential for improvement—as it stands, the Rocket Drawer uses a single Rocket object to store all values. But in reality, several subsets of those values are parts of the Rocket. In the future, I’d like to revamp the object hierarchy and introduce sub-objects into the Rocket object with individual draw methods and then use the main Rocket’s draw method to call the parts’ draw methods. Or possibly even store the Rocket parts as an array of objects to be stored within the Rocket array and then use a foreach loop to call the individual methods.

With Ex3.2, a novel exercise which introduces user input, the concept was simple enough. The complication was self-introduced—I wanted to introduce a vector system based on magnitude and angle rather than component vectors. While in theory it was fairly simple, I forgot I’m not working on a calculator but rather a computer. The Y axis is inverted, which I did initially account for, and the Math.trig functions accept arguments in radians, which I did not account for. I also mangled the actual draw method due to misplaced variables in a translate and actual rectangle draw call. The aforementioned radian error was easily fixed, but the mangled variables were a result of improper planning and poor variable naming. I used internal variables within the draw method that were titled x and y which corresponded to the center of the Rectangle object, but I also had Rectangle.x and Rectangle.y which corresponded to the position of the Rectangle object. Troubleshooting took a second, but the errors were corrected. Notably, the p5.js push() and pop() methods were also self-introduced here to limit issues with calling multiple objects’ draw methods.

With Ex3.3, the criteria for the exercise could have been accomplished as simply as creating an object literal for the ball and possibly velocity to Ex1.4. Another issue was that I did not do Ex1.4, but rather Ex1.3. While I could have done the same to Ex1.3, Ex1.4 evolved with OOP seemed far more interesting. Ex3.3 refines previous functionalities, specifically the random color fade using p5’s lerpColor interpolation function as seen in Ex2.2 Red Remover, as well as Ex3.2’s Vector concepts. Capabilities afforded by OOP also allowed for a more readable canvas object literal which encapsulated canvas width, height, and bounds. The lerpColor implementation in Ex2.2 used an inputColor, outputColor, and an intermediateColor. The inptermediateColor would use lerpColor to approximate a value between the inputColor and outputColor, and then overwrite the existing input color and then proceed to the next loop. Overwriting p5 Color Objects also required extensive decoding the color object into an array and encoding back into a p5 color object. Repetitive approximations also required fuzzy checks for cycle completion. In Ex3.3, I followed Professor Faas’s suggestion of using the actual Intermediate Color as the display color and incrementing the mixture progression parameter. This worked far better. I also went with an OOP approach as I wanted to encapsulate the incremented variables inside the object itself rather than as global variables. I also internalized the parse color function into the Color Fade class. The next critical class was the Vector class which I used a contemporary method of establishing using the ‘class’ keyword and a constructor function. I also used a static method as an alternate way of building a Vector object. The primary Vector object constructor utilizes a directed magnitude whereas the secondary ‘constructor’ uses component vectors along the x axis and y axis. Since the object stores information for both the directed magnitude approach and the component vector approach, I also created ‘rebuild’ functions that rebuilt any affected attributes upon changing one attribute. This represents the greatest area for improvement in this exercise. In the future, I’d like to use official setters and getters to moderate this information flow and to limit any issues with direct variable assignment. Other than that, this lab went excellently. The circle class was also fairly easy to program. As a finishing touch, I went through and found some JS code I used last semester to dynamically generate multiple objects on runtime using properties of the parent ‘window’ object. With this in mind, I then repeated actions in both the setup() and draw() functions with a for loop to repeat the actions to several circle objects and their corresponding attributes which resulted in a beautiful barrage of chaotic ball bounces. In the future, I might like to do a collision system between the balls, however, I suspect it’s more difficult and resource-intensive than I imagine.