University of California, Santa Cruz

Experiment 3:

Generative Methods

Dominic Berardi & Nathan Prieto

CMPM 169: Creative Coding

Modes

Due Date 1/31/2023

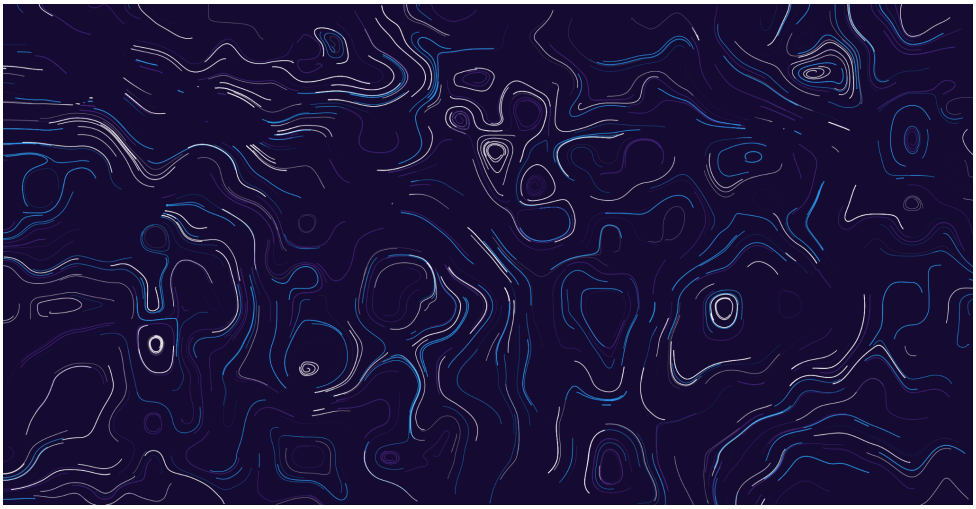
Fireworks

Uses generative methods of Perlin noise and randomness to create colorful, confined firework-like art.

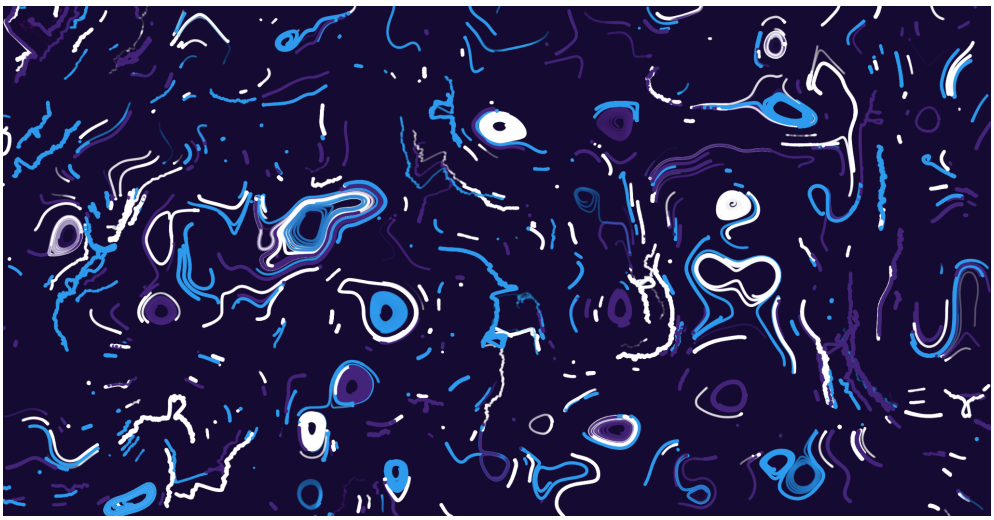
Playable link:
https://ncprieto.github.io/Experiment-3-CMPM-169-Fireworks-Nathan-Prieto-Dominic-Berardi/
Sketch.js code: https://github.com/domberardi10/cmpm169/blob/master/experiment3/js/sketch.js

Step 1: Imitate

We had talked about what examples we liked in class, and settled on this Perlin noise work: https://openprocessing.org/sketch/494102. To start the imitation process, we simply did the most straightforward solution: paste the code word-for-word. Will it translate to p5 from normal Javascript cleanly?



Safe to say, it looks the exact same! What if you change a few values? In this case, we pinpointed where it draws each line as an ellipse() function, and added to what the current radius is.



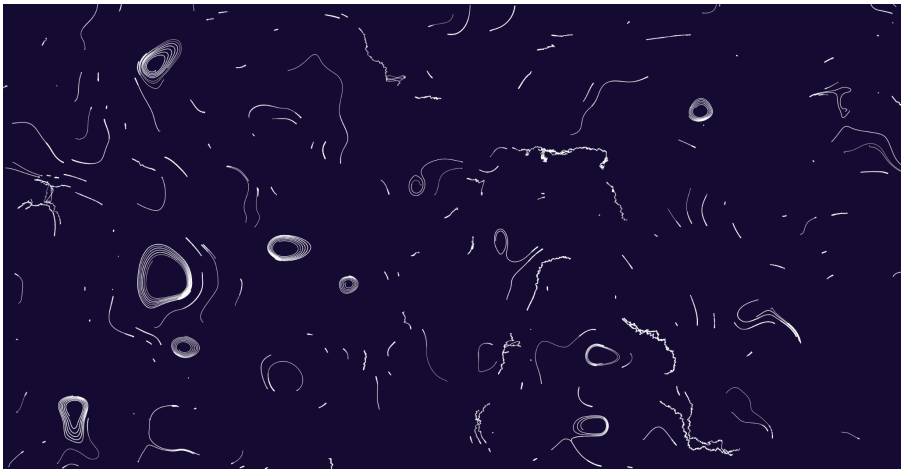
But of course, we're here to learn and imitate, not blindly copy and barely change code. Firstly, we must understand *how* the code works in the first place. At a glance, it is like OOP (but an odd JS version of it), with something called "particles"; these represent each individual line in the piece. Every generated particle is placed into an array of similarly colored particles, which are then iterated through every time the draw() function runs. Each iteration on a particle updates a few things, of which are functions of the particle object: the point is moved inside the code, which is then displayed onto the screen, and then a check to keep everything on the screen ends the run of each particle. An individual particle has vectors tracking its direction, position, and velocity. The noise() function in this code utilizes Perlin noise to choose a new direction for a point to move towards, to which the velocity becomes the direction times the speed, and the new position is that velocity added to the existing position. Lastly, the actual "lines" or "particles" on screen are not actually true lines at all; they are small ellipses.

We noticed that, since this artwork was created using processing, the "particle" object appeared odd and not like an object would be written like using p5. Instead of being a class, Particle was a function, with more functions stemming off of that, such as the ones that move and display each particle. The first order of business for recreating this code in p5 was rewriting the Particle function to be more like the type we'd normally see in p5. The left image is the original code, the right image is our OOP Particle class code.

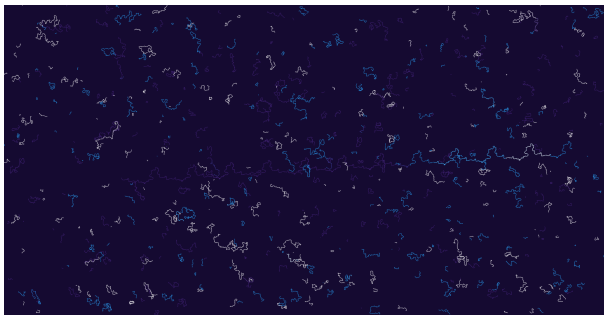
```
function Particle(x, y){
  this.dir = createVector(0, 0);
  this.vel = createVector(0, 0);
  this.pos = createVector(x, y);
  this.speed = 0.4;

  this.move = function(){
    var angle = noise(this.pos.x/noiseScale, this.pos.y/noiseScale)*TWO_PI*noiseScale;
    this.dir.x = cos(angle);
    this.dir.y = sin(angle);
    this.vel = this.dir.copy();
    this.vel.mult(this.speed);
    this.pos.add(this.vel);
  }

  this.checkEdge = function(){
    if(this.pos.x > width || this.pos.x < 0 || this.pos.y > height || this.pos.y < 0){
      this.pos.x = random(50, width);
      this.pos.y = random(50, height);
    }
  }

  this.display = function(r){
    ellipse(this.pos.x, this.pos.y, r, r);
  }
}
```

```
class Particle{
    constructor(x, y, color, alpha, radius, speed){
        this.direction = createVector(0, 0);
        this.velocity  = createVector(0, 0);
        this.position  = createVector(x, y);
        this.speed  = speed;
        this.color  = color;
        this.alpha  = alpha;
        this.radius = radius;
    }

    move(){
        var angle = noise(this.position.x/noiseScale, this.position.y/noiseScale)*TWO_PI*noiseScale;
        this.direction.x = cos(angle);
        this.direction.y = sin(angle);
        this.velocity = this.direction.copy();
        this.velocity.mult(this.speed);
        this.position.add(this.velocity);
    }

    checkEdge(){
        if(this.position.x > width || this.position.x < 0 || this.position.y > height || this.position.y < 0){
            this.position.x = random(50, width);
            this.position.y = random(50, height);
        }
    }

    draw(){
        smooth();
        fill(this.color);
        ellipse(this.position.x, this.position.y, this.radius, this.radius);
    }
}
```

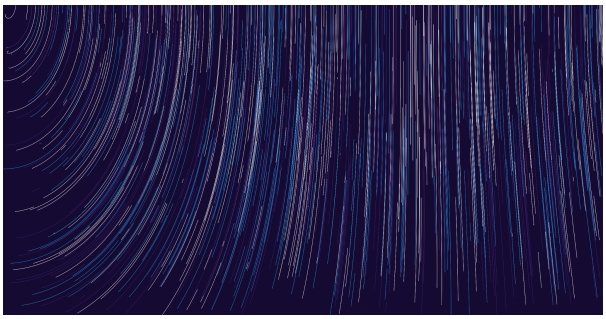
Not many changes to even make to get this working as it should in p5! One specific change we made for clarity's sake is when the colors are determined for each particle; the original implementation had the fill() be changed before each particle was drawn, but now the color is part of the Particle class. We also had one small hiccup where the speed of the ellipses being drawn was not set up in the class constructor properly, which made an interesting result where some lines worked, and some went crazy.



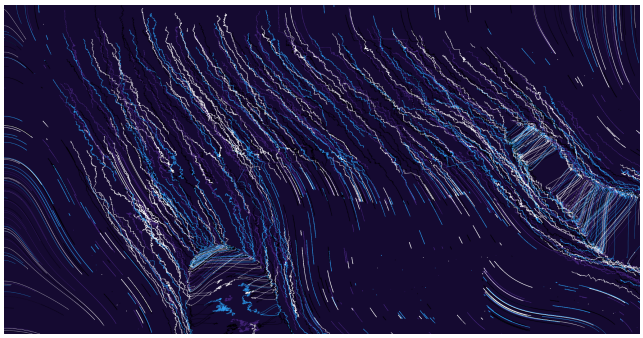
After getting the p5 OOP version to work properly, we played around with the values of stuff to ensure nothing breaks too badly.



Lowered noise scale to 10.



Upped noise scale to 10,000.

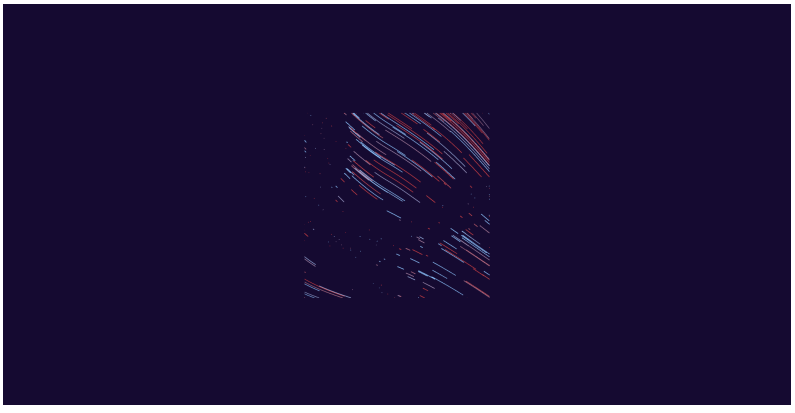


Velocity multiplied by 2.

Things are looking interesting; let's make it even better.

Step 2: Integrate

To begin the integration step, we weren't initially sure what to do. Thus, we looked at the class slides for ideation, and decided it would be best if we tried to integrate a few different basic p5js generation functions into what we already had: things like random(), lerp(), constrain(). The first thing we began to try out is the constrain() function. It would be interesting to attempt to fit the particles we generated into certain boundaries. The first test involved setting up the constrain() function to ensure it would even work how we wanted it to. The result was… alright. Some of the particles weren't quite moving, though.

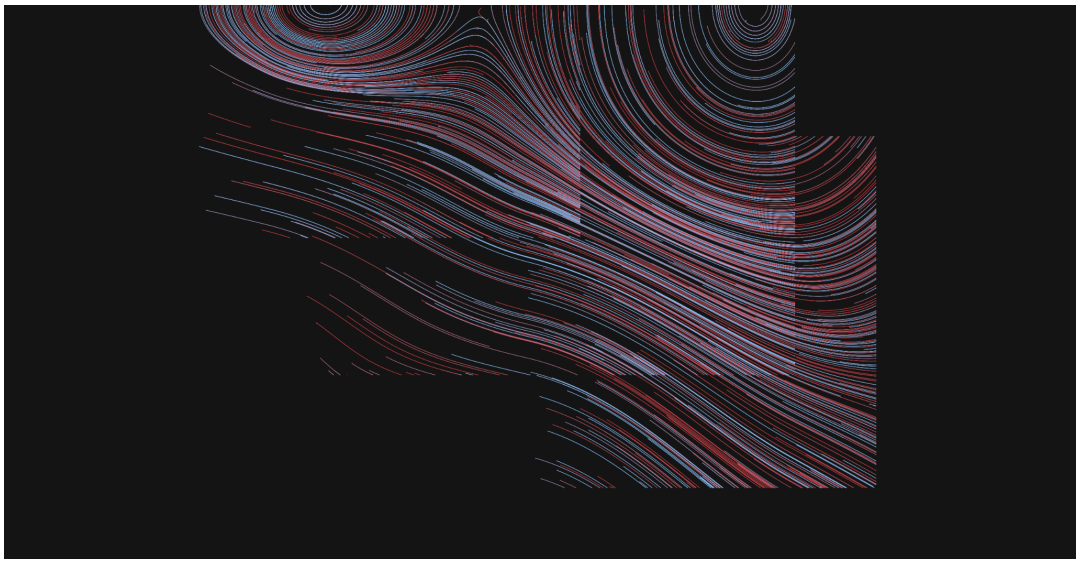


```
move(){
    this.position.x = constrain(this.position.x, this.centerX - this.squareSize, this.centerX + this.squareSize);
    this.position.y = constrain(this.position.y, this.centerY - this.squareSize, this.centerY + this.squareSize);
```

We thought that these lines may not be moving as they think they are outside the boundary. In an attempt to debug, we ended up ruining the original boundary check code, as well as not realizing the square's center was being randomized and applied to every separate particle. This resulted in a whole lotta dots.

We realized that we would need some sort of parent-like object for the particles, a square shape that contains information about the square itself– previously, all of the information was a mess of variables. The Shape class was created, and squares of particles could be placed within the canvas.

```
class Shape{
    constructor(){
        this.centerX = random(500, windowWidth  - 500);
        this.centerY = random(500, windowHeight - 500);
        this.size = random(200, 500);
        this.array = [];
        this.createArray();
    }

    createArray(){
        for(var i = 0; i < nums; i++){
            this.array[i] = new Particle(random(this.centerX - this.size, this.centerX + this.size),random(this.centerY - this.size, this.centerY + this.size),
            1, 1, 0.4, noiseScale, this.centerX, this.centerY, this.size);
        }
    }

    draw(){
        for(let i in this.array){
            this.array[i].draw();
        }
    }
}
```

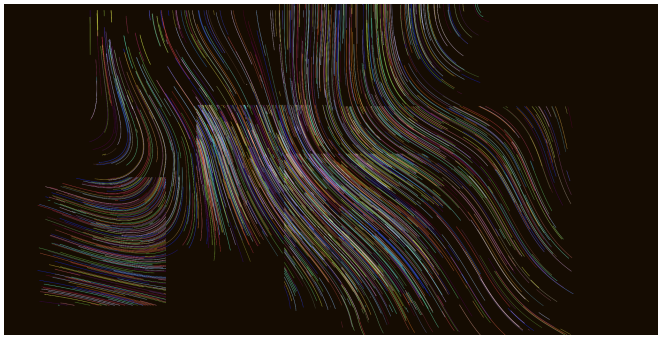


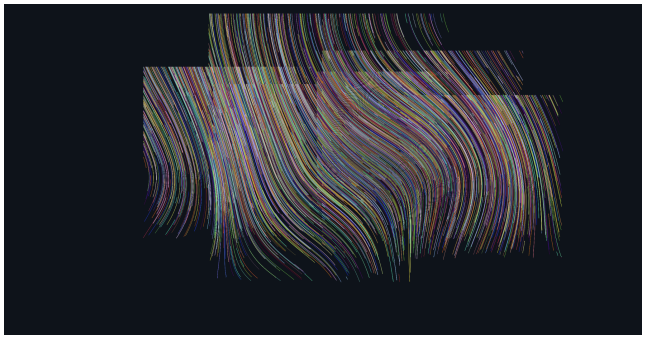
Looking great, but things still feel a bit too similar to the original piece. How about more colors? Make an array of a bunch of random colors, choose two to mix at varying amounts? Infinite colors! This was a good time to use a lerp() function. Except, colors don't take the regular lerp, they're special and get a function called colorLerp(). Apply it to the background too, why don't ya.

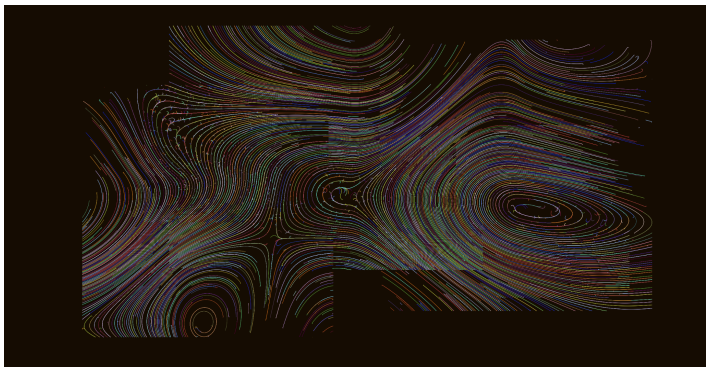
```
this.color = lerpColor(color(color_array[int(random(0, 10))]), color(color_array[int(random(0, 10))]), random(0, 1));
```

```
background(lerpColor(color(0, 0, 0), color(color_array[int(random(0, 10))]), 0.1));
```

For the last part of this step, we increased the amount of squares on the screen.







Step 3: Innovate

For the final step of the experiment, we knew we wanted to try adding fractals to what we've already created. But this proved much harder than expected. Here's why: fractals rely on creating the lines they make in the moment the function runs; our particles, on the other hand, are drawn before and during when a fractal branch() function would have to be called. Attempting to take the classic fractal creating function and adapting it to our needs didn't go so well. Below is the sad code that couldn't be resulting from that:

```
// function branch(p) {
    //p.angle *= 0.67;
    //delta += deltaTime / 1000;
    //if (delta >= 5){
        //push();
        //rotate(p.angle);
        //branch(p);
        //pop();
        //push();
        //rotate(-p.angle);
        //branch(p);
        //pop();
        //delta = 0;
    //}
//}
```

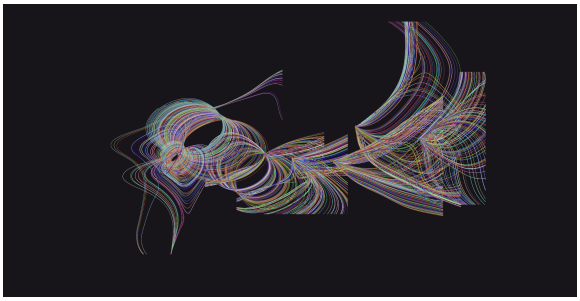
```
// if(delta >= 1.5){
//     let factor = int(random(0, 2)) == 0 ? -1 : 1;
//     this.array.push(new Particle(this.array[i].x, this.array[i].y, 1, 1, 0.4, noiseScale - (random(200, 500) * factor), this.centerX, this.centerY, this.size));
//     delta = 0;
//     this.array.splice(0, 1);
//     console.log(this.array);
// }
```

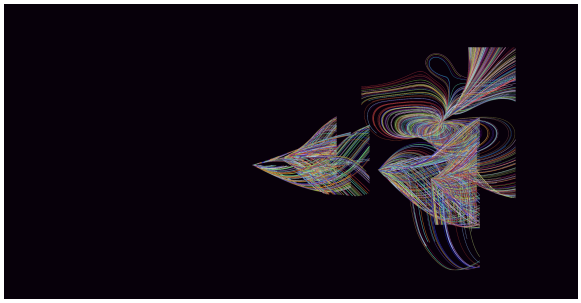
After fractals didn't go so hot, we were down on ideas. A couple of things we're tried, including using timers, also weren't turning out so good. But we noticed something odd about a few of our squares; they had many particles emanating from the square's origin point. It looked quite cool (no image, sorry) so we decided to see if we could harness what was happening there into something more intentional.

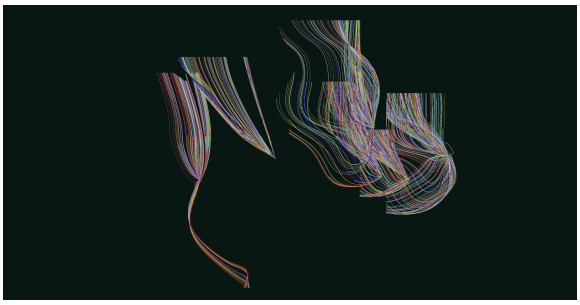
Looking at our code extensively for *why* the stemming from one point was happening didn't turn up anything. What if we forced the particles to stem from a certain point? What if that point was in the center of each square? Well, we have that info!

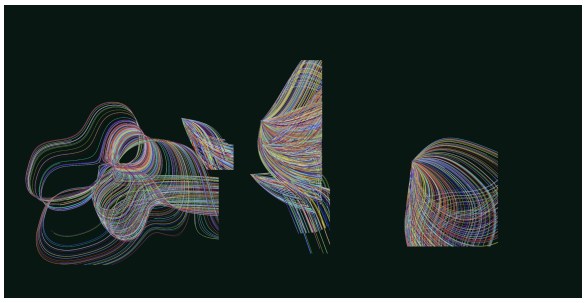
```
createArray(){
    for(var i = 0; i < nums; i++){
        let factor = int(random(0, 2)) == 0 ? -1 : 1;
        this.array[i] = new Particle(this.centerX, this.centerY, 1, 1, 0.4, noiseScale - (random(500, 1500) * factor), this.centerX, this.centerY, this.size);
    }
}
```

Not too much code added, but after trial and error and fractalized sadness, we made something awesome.









Reflect

Dominic:

For this assignment, I decided two things: one, that I would work with a partner (to which I have the talented Nathan Prieto) and that we would not make the mistake I had of the last assignment, where I did not start with an existing processing work to emulate. Nate had already worked a little bit with Perlin noise, but I have not; joining together would hone his skills and teach me. To effectively pair-program, we utilized the Live Share extension for VSCode, which allowed us to essentially be pilots at the same time. Though, I will admit that Nate did more coding than I, but I was documenting and writing much of it. I feel like we were still balanced in how hard we worked. The best part about this experiment was simply being able to talk with another person about things we were stumped on. The worst part was having to concede to not creating the fractals we would've wanted.

Nathan:

I had a really great time working with Dominic on this project. This was my first time really working on a collaborative art piece with a partner and felt as if our combined efforts made something really unique. Using Live Share for VSCode is what really made this project possible as we could both work on it simultaneously without having to use any sort of source control or file sharing which was really convenient. Refactoring the existing code to create something that was more object oriented and in line with modern JS standards was a great step in developing skills surrounding the idea of object oriented programming. Creating code that is object oriented allows for clarity of code, reductions in redundancy, and overall faster development times which helped in creating this project.

Self-Evaluation

Dominic

Self Evaluation Rubric						
Did you complete the assignment and did you complete it on time?	**Submitted on time**	**Up to 1 day late**	**Up to 2 days late**	**Up to 3 days late**	**4 days late or more**	**Do you need to clarify?** **No**
	X	☐	☐	☐	☐	
Did you collaborate with a partner?	**Worked with partner**			**Worked alone**		**Do you need to clarify?** **No**
	X			☐		

Did you put in earnest effort and provide an articulate summary of your experience?	**Excellent**	**Pretty good**	**About average**	**Could be improved**	**Not this time**	**What supports this?** **I tried to explain my process as thoroughly as possible. Used many pictures.**
	X	☐	☐	☐	☐	
Was the assignment complete, with minimal errors, correct output, and good style?	**Excellent**	**Pretty good**	**About average**	**Could be improved**	**Not this time**	**What supports this?** **Had two people look over each part of the assignment.**
	X		☐	☐	☐	
How much EXTRA effort did you put into the assignment?	**A lot of extra effort**		**Some extra effort**		**Not this time**	**What supports this?** **Just wanted it to be done after a while.**
	☐		**X**		☐	
Reflection in section above.						

Nathan

Self Evaluation Rubric						
Did you complete the assignment and did you complete it on time?	**Submitted on time**	**Up to 1 day late**	**Up to 2 days late**	**Up to 3 days late**	**4 days late or more**	**Do you need to clarify?** **No**
	X	☐	☐	☐	☐	
Did you collaborate with a partner?	**Worked with partner**			**Worked alone**		**Do you need to clarify?** **No**
	☐			**X**		
Did you put in earnest effort and provide an articulate summary of your experience?	**Excellent**	**Pretty good**	**About average**	**Could be improved**	**Not this time**	**What supports this?** **Our process is well documented with adequate pictures that show every step of the project.**
	X	☐	☐	☐	☐	
Was the assignment complete, with minimal errors, correct output, and good style?	**Excellent**	**Pretty good**	**About average**	**Could be improved**	**Not this time**	**What supports this?** **The writeup is sufficient enough for what the teaching**

	X		☐	☐	☐	team expects and our code is neatly organized.
How much EXTRA effort did you put into the assignment?	A lot of extra effort		Some extra effort		Not this time	What supports this? We really put in the honest effort into creating code that adheres to modern programming practices. Our project is certainly unique but definitely has room for extra features.
	☐		X		☐	
Reflection in section above.						