Connor Case
MART 340 Final Project Paper

Leonardo da Vinci once said that “Art is never finished, only abandoned.” It’s a quote that has always resonated with me, and whether or not one agrees with the implications of it, it cannot be denied that, in most traditional cases, a work of art becomes stagnant once its artist ceases to work on it. This doesn’t necessarily mean that it is incapable of motion; works of art such as films certainly contradict that, but it does often mean that the experience of viewing it will no longer change.

For my final project, I set out to create something that would continue to provide different experiences even after I was done working on it. This was to be a piece of moving abstract visual art that would be randomly generated and change randomly over time. The goal was for it to continuously create new visual patterns as it updated and moved.

One of the most difficult parts of making this project was the act of beginning my work on it. Unlike previous projects I have worked on, this one did not have a set of specific, defined features to be created. Instead, I needed to figure out a way to design something that had the word “random” as a key component of its description. Needless to say, it took me a little while to get this off the ground.

A key component of the project that I began with was ensuring that I could actually create shapes to fill my sketch with. I first created a function to make a quadrilateral shape that would be generated randomly within an area slightly larger than the sketch. I created a number of functions to allow it to do a number of things that I needed it to do. I then created similar pages of code that could generate other types of shapes, triangles and ellipses, in similar ways.

The first step was very simple; drawing the shape. I created a number of different variables that controlled the dimensions and color of the respective shapes, and used them to define the values in the drawShape function. This allowed me to set them to random values when the sketch was run, as well as letting me change them.

```
this.ellipseX = random(canvWidth); // variables for the ellipses
this.ellipseY = random(canvHeight);
this.ellipseWidth = random(canvWidth);
this.ellipseHeight = random(canvHeight);

this.ellipseXChange = random(-5, 6); // variables to change ellipse positions
this.ellipseYChange = random(-5, 6);
this.ellipseWidthChange = random(-5, 6);
this.ellipseHeightChange = random(-5, 6);

this.rFill = random(256); // randomizes color fill
this.gFill = random(256);
this.bFill = random(256);

this.rFillDirec = random(this.trueFalse); // used to change colors
this.gFillDirec = random(this.trueFalse);
this.bFillDirec = random(this.trueFalse);
```

I created functions that would change the values of these respective variables by adding the values of other variables I had created to them. That would allow the dimensions of the shapes and their color fills to change. By making them change through variables as well, I could also go on to change those variables, meaning that the values that were changing the values that were affecting the visible shapes were being randomly changed. This allowed me to keep the shapes and colors moving smoothly, while still having their values be rooted in randomness.

```
MakeQuad.prototype.changeColor = function() {
  this.rFill += this.rFillDirec; // increases color
  if (this.rFill >= 255 && this.rFillDirec > 0 || this.rFill <= 0 && this.rFillDirec < 0) {
    this.rFillDirec = this.rFillDirec * -1; // resets it to 0 if it gets to 255
  }
  this.gFill += this.gFillDirec; // increases color
  if (this.gFill >= 255 && this.gFillDirec > 0 || this.gFill <= 0 && this.gFillDirec < 0) {
    this.gFillDirec = this.gFillDirec * -1; // resets it to 0 if it gets to 255
  }
  this.bFill += this.bFillDirec; // increases color
  if (this.bFill >= 255 && this.gFillDirec > 0 || this.bFill <= 0 && this.gFillDirec < 0) {
    this.bFillDirec = this.bFillDirec * -1; // resets it to 0 if it gets to 255
  }
}
```

Next, I had to control when these various changes would happen. In order to do this, I created timers that would set a random frame goal and count up to it. When it was reached, they would execute a function and set a new random goal. This function is what would go about changing the Change variables, so that not only would the way movement changed be random, but the time it changed at would be as well.

```
MakeTriangle.prototype.timer = function() { // creates a timer
  if (this.secTime >= this.timeGoal) {
    this.shapeChangeChange(); // calls function when timer elapses
    this.secTime = 0; // resets timer
    this.timeGoal = random(0, this.secTimeMax + 1); // resets random timer goal
  } else {
    this.secTime++; //increments timer
  }
}
```

I also had to ensure my shapes didn't wander too far away from the canvas, so I created functions that would detect when they moved a certain amount past the edge (I picked 200 pixels) and reversed the x or y movement values of a point, respectively.

Of course, all of that was just to ensure that the shapes were able to be generated and move properly. I then had to move to the main sketch itself to write the code that would handle actually creating multiple instances of these elements.

To make multiple instances of these multiple types of shapes, I had to make nested arrays. I made an array called "shapes" that contained three other arrays, "quads," "ellipses," and "triangles." I used for loops in the setup function to fill these smaller arrays with shapes, and then used a nested for loop in the draw function ensure all the necessary functions in these elements were being repeatedly called for each individual shape in each shape group.

```
  shapes = [quads, ellipses, triangles]; // assigns the type of shape to an array value
}

function draw() {
  background(255, 1);

  for (var j = 0; j < shapes.length; j++) { // continuously draws all shapes and calls functions
    for (var i = 0; i < shapes[j].length; i++) {
      shapes[j][i].drawShape();
      shapes[j][i].changeSize();
      shapes[j][i].timer();
      shapes[j][i].changeColor();
    }
```

I also wanted new shapes to enter the sketch, as well as for the old ones to disappear. I accomplished this through creating two more timers, each of which did specific things to the arrays. The first of these timers, when triggered, would call a function called newShape. This would be called with arguments that would select a random one of the shape types, create a new shape of that type, and append it to the end of its specific array.

```
function newShape(shapeArray, shapeType) { // need the two values to run
  var newShape;

  if (shapeType == 0) {
    newShape = new MakeQuad(windowWidth, windowHeight); // makes a new quad
  } else if (shapeType == 1) {
    newShape = new MakeEllipse(windowWidth, windowHeight); // makes a new ellipse
  } else if (shapeType == 2) {
    newShape = new MakeTriangle(windowWidth, windowHeight); // makes a new ellipse
  }

  shapeArray[shapeType].push(newShape); //pushes the newly created shape into the array
}
```

The second of these timers, when triggered, would use the shift command to remove the earliest value of a random one of the arrays; effectively removing an old shape. These two timers being separate further increased the fluidity and unpredictability of the sketch, as they will trigger at different times.

Finally, I wanted the sketch to be able to be viewed in a number of different ways, and so I made sure that it would be able to adapt to changes in window size. I used the windowResized function to send new window width and height values to their respective locations every time the size of the window changed, and I added functions inside the code of the objects that would ensure they would update their values to accommodate for these changes. Now, when the window is resized, the shapes will move to fill the empty space or compress to fit in the smaller area.

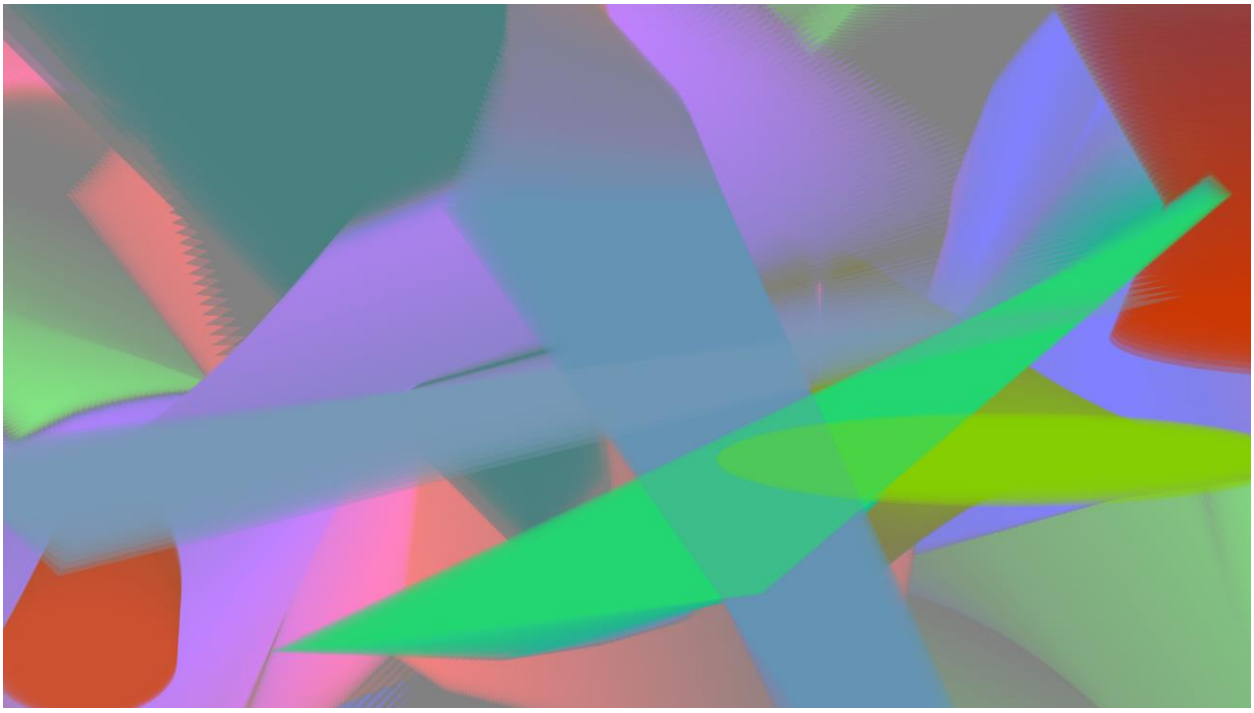
I also added a mousePressed function to bring the sketch into a borderless fullscreen, and set the cursor to be invisible whenever it is on top of the sketch. This allows a user to make this program completely fill up a monitor, and use it as a display of art, if they wish.

```
function windowResized() {
  resizeCanvas(windowWidth, windowHeight); //changes the canvas size

  for (var j = 0; j < shapes.length; j++) {
    for (var i = 0; i < shapes[j].length; i++) {
      shapes[j][i].resizeWindow(windowWidth, windowHeight); //calls resizeWindow for all shapes
    }
```

When the project is run in practice, its visual appearance will be entirely random, and it will be in motion, so it is difficult to present an idea of what it looks like with a still image. Still, one example of something that it can generate is shown here.

Connor Case
MART 340 Final Project Paper



All in all, this was a very interesting project to work on. It was a big step away from many of the more concrete, goal-oriented problems we tackled earlier in this class, and because of that I feel it was a valuable experience. I am unsure if I will ever be able to expand upon it in the future, and it may not be anything revolutionary, but regardless, I am proud of it.