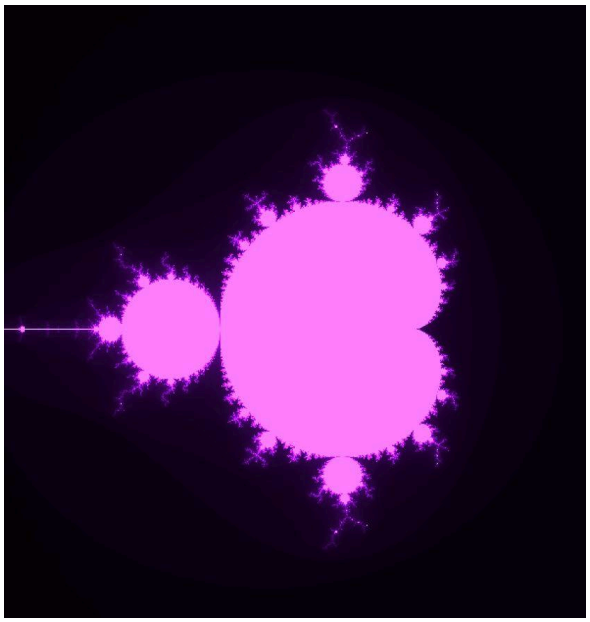
Introduction to Mandelbrot Set Methods, by Ann Song. March, 2024. For Robotics And Beyond AI Initiative series of workshops.

I. Introduction to Generative Art and Processing (10 minutes)

The slides created for the Part 1 workshop are at this link. Reviewing these slides will help prepare for the next lesson (Part 2) below. The slides give a good introduction to Generative Art and what mandelbrot sets are and how they can be used to create artistic designs using mathematics and computer-based algorithms.
Gen Art Part 1 Slides with links

II. Introduction to Mandelbrot Set Methods

This outline describes the mathematical concepts needed to create Mandelbrot sets and how they are used.



How do we create this? Well it's actually quite simple.

To create an image like above, we start with making a map of three rows of squares.
This is called a grid or an array. Programmers call it a Coordinate Plane.
This is like making a graph in your math class, with an X and Y axis.
Think of each square as one pixel. A pixel is the name for the smallest block of color in an image on the screen of a computer, tablet or phone.

We give each block a code number, like a 1, 2 or 3, but we decide which blocks have which number with a secret formula

Then we pick a color for each number.

The Secret Formula

Ok so what's this mysterious formula that unlocks this world of art?

Well we start with a grid. The grid is actually called a complex coordinate plane.

But what's a complex coordinate plane?

It's complicated. It involves a concept in math called imaginary numbers. We don't have time in this workshop to explain imaginary numbers but we will use them here.

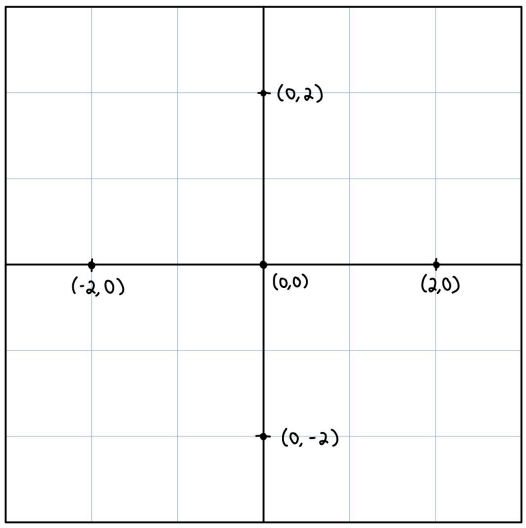
To help you understand the "secret formula," we will use a regular coordinate plane, or grid, and explain the difference.

Here are the steps for making the formula.

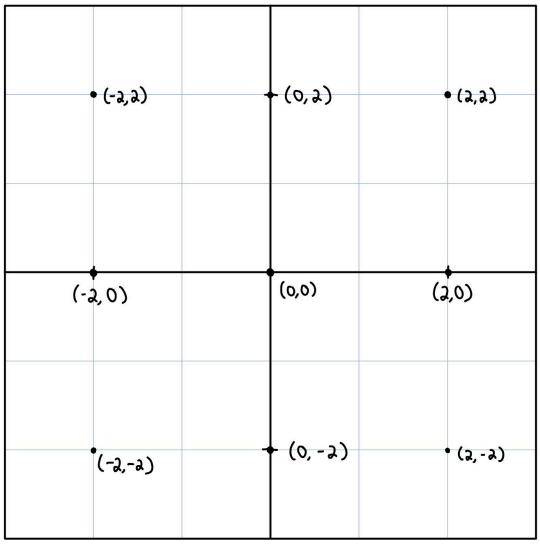
1. Make a map of your grid, or Coordinate Plane.
2. Choose a starting point on the coordinate plane
3. Perform *The Secret Formula* (we will talk about this later)

Step 1

Let's map out a coordinate plane that goes from -2 to 2 on the x and y axis.



Let’s add the points on the corners too, so we know where they are.



Now we will change these points into cells so the grid, or Coordinate Plane is more simple to look at. This is the final diagram we will use.

-2, 2	0, 2	2, 2
-2, 0	0, 0	2, 0
-2, -2	0, -2	2, -2

Step 2

To simplify the grid, we will now go through each of these cells and perform *The Secret Formula* and replace each set of points with a single number. Let's take (2, 2), the top right, as an example and assign it to **c**. We will also define a variable **n** that is equal to 1 and a variable $\mathbf{z_n}$ that will be equal to the starting value of **c**.

Starting Values:

$c = (2, 2)$ **This is the area of the upper right square of our grid.**

$n = 1$ **"n" is a counter so we can know how many iterations we have done. An iteration is one cycle through the equation.**

$z_n = (2, 2)$ **We will get into "z" more in a bit. For now, it is always the initial value of "c"**

$zx = 2$ **x coordinate of "z"**

$zy = 2$ **y coordinate of "z"**

limit = 4 **An arbitrary limit**

We will now use these starting values to perform *The Secret Formula*.

<u>The Secret Formula</u>

1. $Check\ if\ (zx)^2 + (zy)^2 \geq limit \rightarrow$ Point C "escapes", return **n** and proceed to the next point (exit The Secret Formula)
2. Increase n by 1
3. $z_n = z_{n-1}^2 + c$
4. If $n \geq 3 \rightarrow$ we give up and just return **n** (exit The Secret Formula)

What did we just do? The goal of this is to come up with a value of **n** for each square. These values of n will then be mapped to colors that create the design. Each value of n will be assigned to a different color.

After the completion of on every point (9 points), we have the following map of n values.

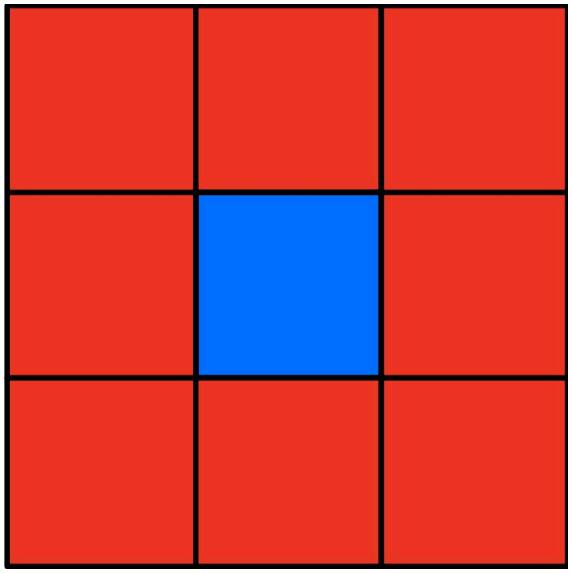
1	1	1
1	3	1
1	1	1

Step 3. Coloring Pixels

We need to think of every cell of our grid as a pixel on a computer screen, or in a photo.
If each cell is a pixel in the above image, then the photo is 3x3 pixels.
Now we want to give each pixel a color so we can make interesting patterns with our secret formula and the computer code we will create.

For each pixel:

1. If its n value $= 1 \rightarrow$ color the pixel red
2. If its n value $= 2 \rightarrow$ color the pixel green
3. If its n value $\geq 3 \rightarrow$ color the pixel blue



Yeah, that's it. That's our photo. Looks boring right? How do we get it to look like the mandelbrot image we saw earlier?

Before we get to that, let's write some code in Processing to get to this point first. After we write this code, minimal changes will be made for it to look like a fully-fledged Mandelbrot set. Here is a link to the code we will write in the workshop. You can look at it before the workshop and it will be useful if you are not attending the workshop.

https://github.com/gmsong06-alt/part2-generative-art-robotics-and-beyond/tree/main/src

Creating a fully-fledged Mandelbrot Set

Cool, so now that the code is done, let's turn it up a notch. We can do this by changing 3 things.

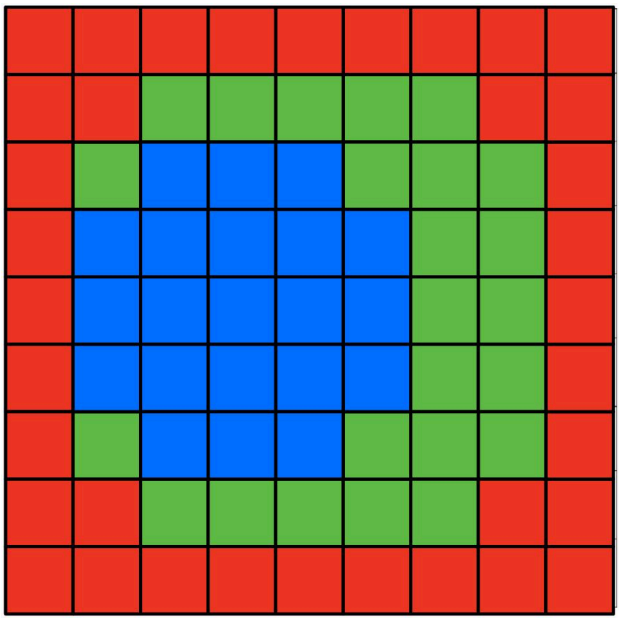
1. The number of pixels in the image
2. When we decide to give up (the maximum value of n)
3. How we color the pixels

Option 1

Alright, let's say we want a 9x9 pixel photo. We start by zooming in on our map of the coordinate plane to get something like this.

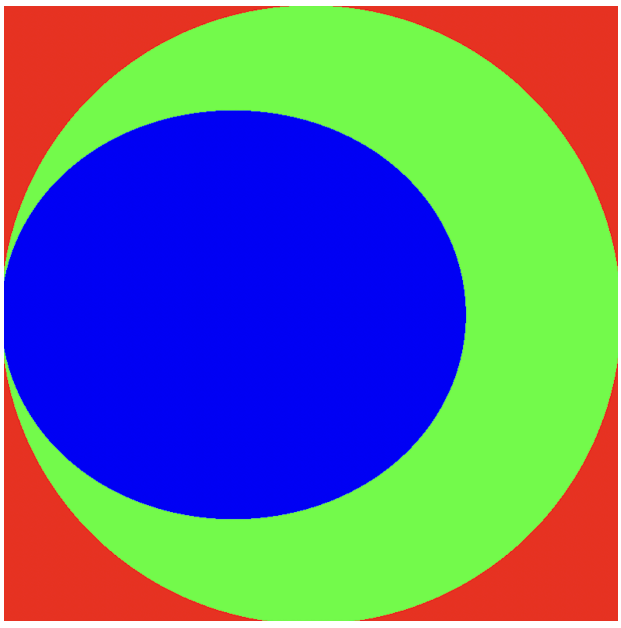
-2, 2	-1.5, 2	-1, 2	-0.5, 2	0, 2	0.5, 2	1, 2	1.5, 2	2, 2
-2, 1.5	-1.5, 1.5	-1, 1.5	-0.5, 1.5	0, 1.5	0.5, 1.5	1, 1.5	1.5, 1.5	2, 1.5
-2, 1	-1.5, 1	-1, 1	-0.5, 1	0, 1	0.5, 1	1, 1	1.5, 1	2, 1
-2, 0.5	-1.5, 0.5	-1, 0.5	-0.5, 0.5	0, 0.5	0.5, 0.5	1, 0.5	1.5, 0.5	2, 0.5
-2, 0	-1.5, 0	-1, 0	-0.5, 0	0, 0	0.5, 0	1, 0	1.5, 0	2, 0
-2, -0.5	-1.5, -0.5	-1, -0.5	-0.5, -0.5	0, -0.5	0.5, -0.5	1, -0.5	1.5, -0.5	2, -0.5
-2, -1	-1.5, -1	-1, -1	-0.5, -1	0, -1	0.5, -1	1, -1	1.5, -1	2, -1
-2, -1.5	-1.5, -1.5	-1, -1.5	-0.5, -1.5	0, -1.5	0.5, -1.5	1, -1.5	1.5, -1.5	2, -1.5
-2, -2	-1.5, -2	-1, -2	-0.5, -2	0, -2	0.5, -2	1, -2	1.5, -2	2, -2

We have essentially changed the intervals from 1 to 0.5 to create a 9x9 board. Performing *Formula A* and our color mapping, the exact same thing as with the 3x3 board, we get this photo.



Let's implement this into the code. Just change the dimensions of the canvas from 3x3 to 9x9.

What if we increase the number of pixels even more? What if we do 1000x1000? Obviously we will not be showing a full 800x800 coordinate plane, so let's just change the dimensions of the canvas again from 9x9 to 800x800. Using 800x800 makes sense because …… ??????



It's better, but still boring. Let's look at Option 2 and 3. These go hand-in-hand.

Option 2 and 3

Let's first change the **maxIterations** variable **"n"** from 3 to 4.

Did anything change? No. Not before we do Option 3, changing the color mapping.

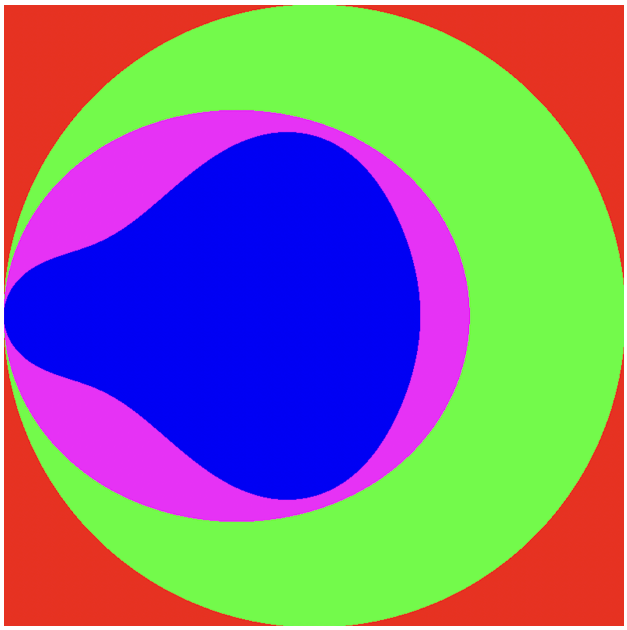
This is our current code:

```
color calculateColor(int n) {
  if(n == 1){
    return color(255, 0, 0);
  }
  else if(n == 2){
    return color(0, 255, 0);
  }
  else{
    return color(0, 0, 255);
  }
}
```

Add another else if so it looks like this:

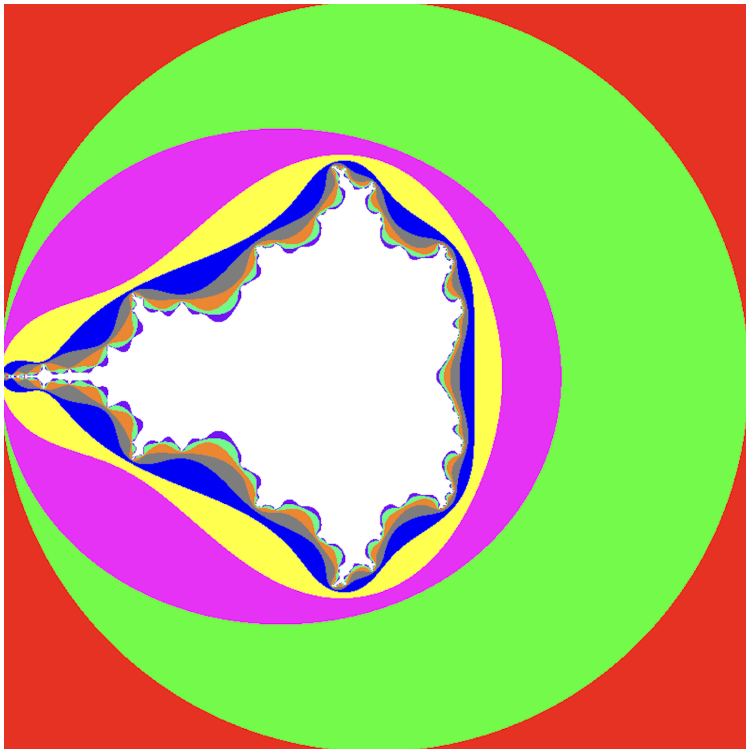
```
color calculateColor(int n) {
  if(n == 1){
    return color(255, 0, 0);
  }
  else if(n == 2){
    return color(0, 255, 0);
  }
  else if(n == 3){
    return color(255, 0, 255);
  }
  else{
    return color(0, 0, 255);
  }
}
```

Now we see some differences.

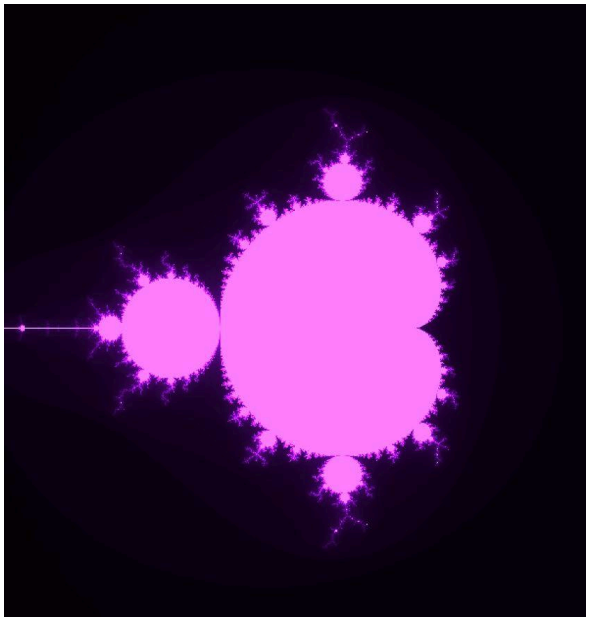


Try changing the **maxIterations** to 10 and adding new else if statements in the code.

As an example, we can end up with something like this.



It looks a lot closer to an actual Mandelbrot set, but it's pretty ugly. How do we create an image like this?



It’s not hard at all.

Here’s what we change.

First, make **maxIterations** equal to 255.

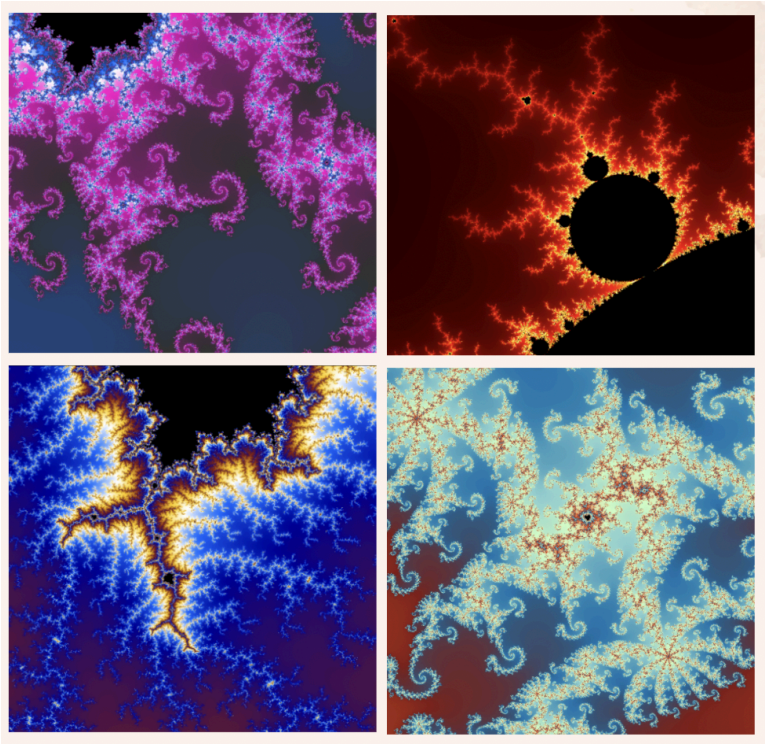
Change the **calculateColor** function to look like this.

```
color calculateColor(int n) {
  return color(n * 4, n/2, n * 6);
}
```

That’s it. Everything else is the same.

Additional Customization

We know how to create the basic shape of the Mandelbrot set now. But how do we customize colors or customize the shape? How do we create elaborate designs like this?

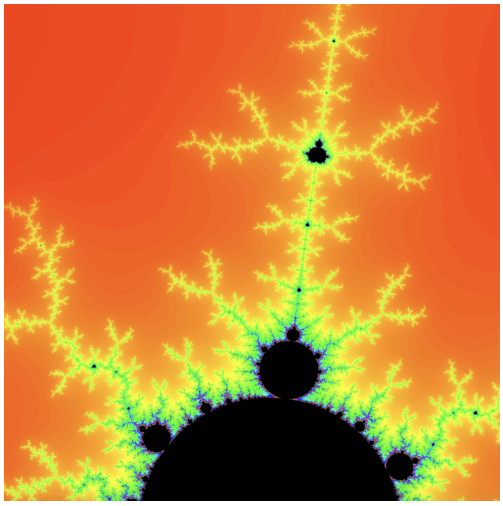


We do this by zooming in and with custom color mapping algorithms.

1. **Zooming in**

This is a Mandelbrot Viewer where you can zoom in on wherever you would like in the Mandelbrot set: https://math.hws.edu/eck/js/mandelbrot/MB.html

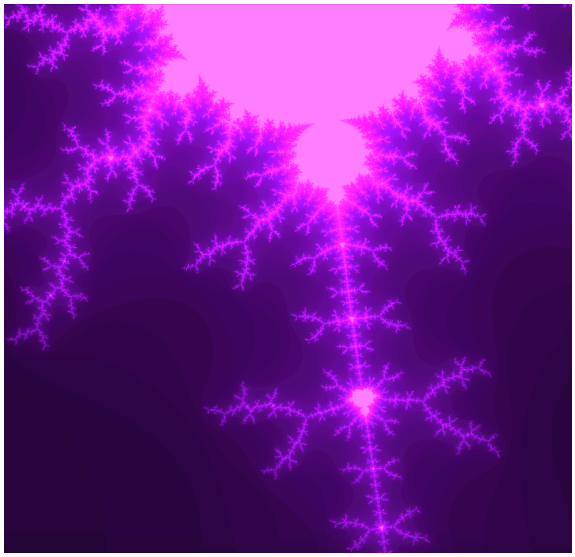
Zoom in on an interesting part of the Mandelbrot image. Don't zoom in too many times as Processing does not have the numerical precision for that. Keep it 2-3 times. Here's an example.



Click on **Show XML Import/Export → Grab Current Example.**

Copy the xmin, xmax, ymin, ymax from the xml (to around 4 decimal places) and replace the -2, 2, -2, 2 with the values you copied.

Run the code and you should see a version of what you saw in the viewer.



Pretty cool right? You can get a bunch of shapes and variations using this method.

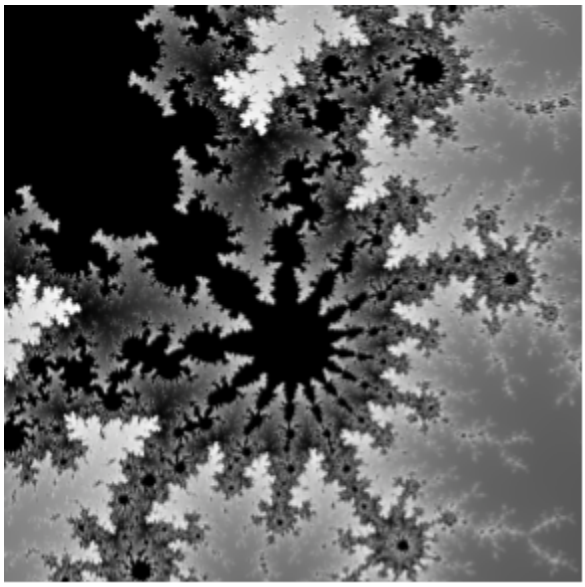
2. Color Algorithms

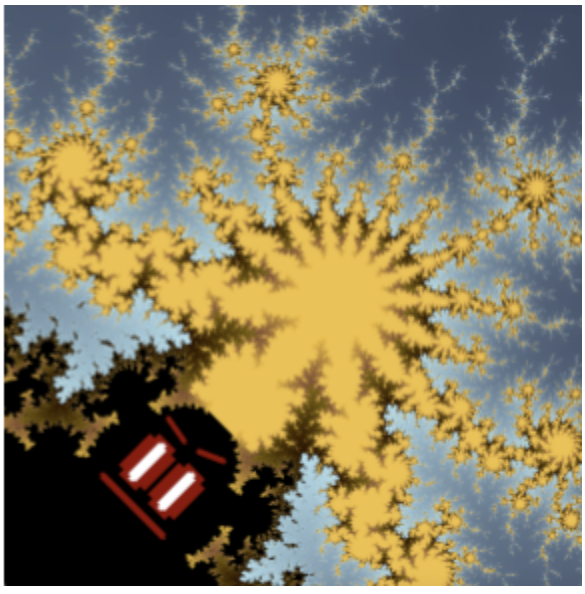
There are so many possible color algorithms you could come up with. It doesn't have to be the current code we have that looks like this.

```
color calculateColor(int n) {
  return color(n * 4, n/2, n * 6);
}
```

Be creative with how you want to color the image. Use noise and random functions or experiment with complex mathematical concepts. Be as simple or complicated as you would like. This is one of my favorite parts of creating Mandelbrot images.

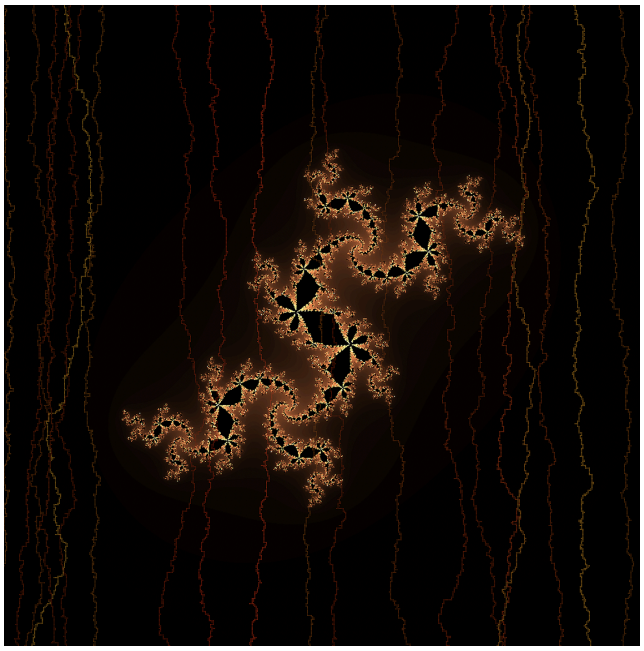
For example, in this image, I was thinking outside the box. The left is the image I first generated, and the right is after I added my custom coloring and a 90 degree rotation.





You can see the resemblance, but the image on the right is a lot different isn't it? That's the power of color algorithms.

There are other types of fractals like the Julia Set, which is connected to the Mandelbrot Set that you could look into as well if you're curious! It has its own algorithm and coloring techniques. Here's an example of a Julia fractal I created:



There are many fractals that can be explored! Here are some links for more information:

Julia Set: https://en.wikipedia.org/wiki/Julia_set
Burning Ship: https://en.wikipedia.org/wiki/Burning_Ship_fractal