Lab Report 1

Connor & Matthew

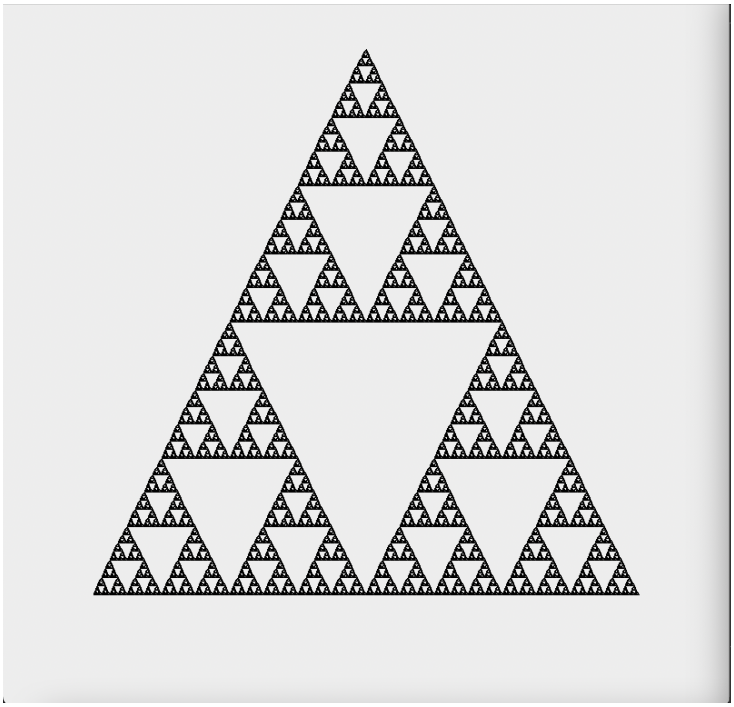
CSC 326 Mathematical Experiments in Computer Science

Professor Simonson

In this lab, we used several elements from within the realm of computer science to create and draw mathematical algorithms. All of the exercises within the lab had us coding logic that allowed us to run the algorithms with specified parameters while making visual models of the algorithms. Being able to use a computer to draw these algorithms is very important because it would normally be nearly impossible to properly draw most of these algorithms by hand. Implementing these algorithms in the form of code is very important for coders to learn because it allows them to demonstrate their ability to translate real math problems into a computer program.

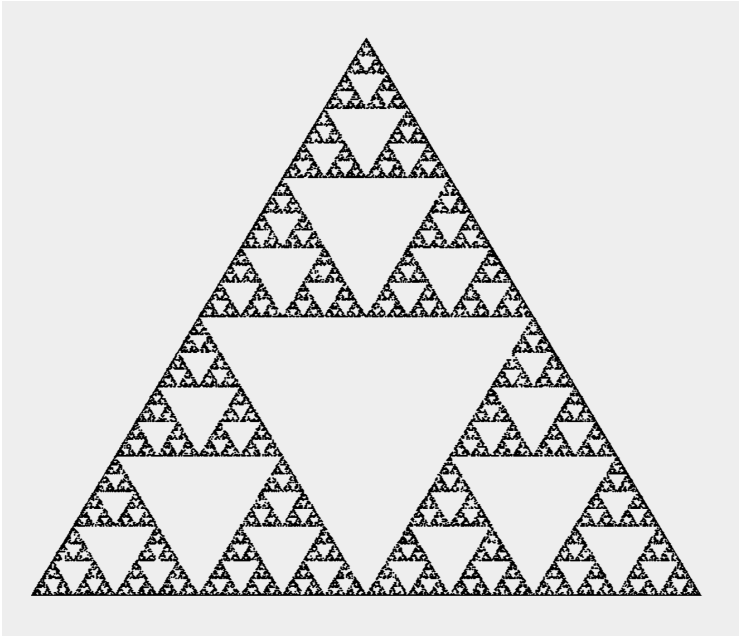
Exercise 1

In exercise 1, we were tasked with drawing Sierpienskis gasket. The idea behind this exercise was that we wanted to demonstrate that we could implement a complex math equation into an arguably more complex recursive program. The program requires user input to decide the depth of the triangle that it is going to draw and works by using three recursive methods that draw triangles within themselves until they reach the desired depth. This program not only demonstrates a knowledge of recursion but it also demonstrates the ability to implement a complex algorithm.



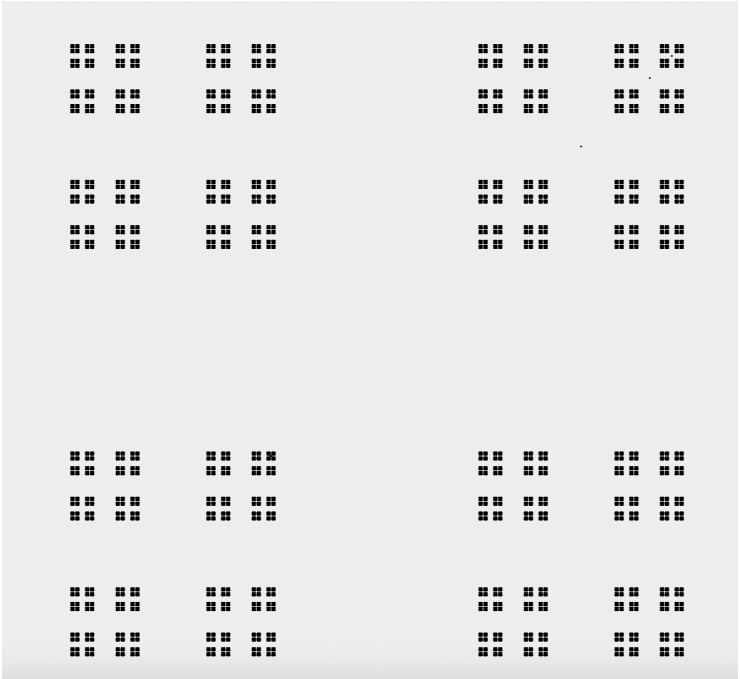
Exercise 2

The purpose of exercise 2 is to create a visual representation of an iterative algorithm that generates a figure through a combination of randomness and predetermined starting conditions. The figure produced by this algorithm is a visual representation of how randomness and predetermined conditions interact to create a structured yet unpredictable pattern.



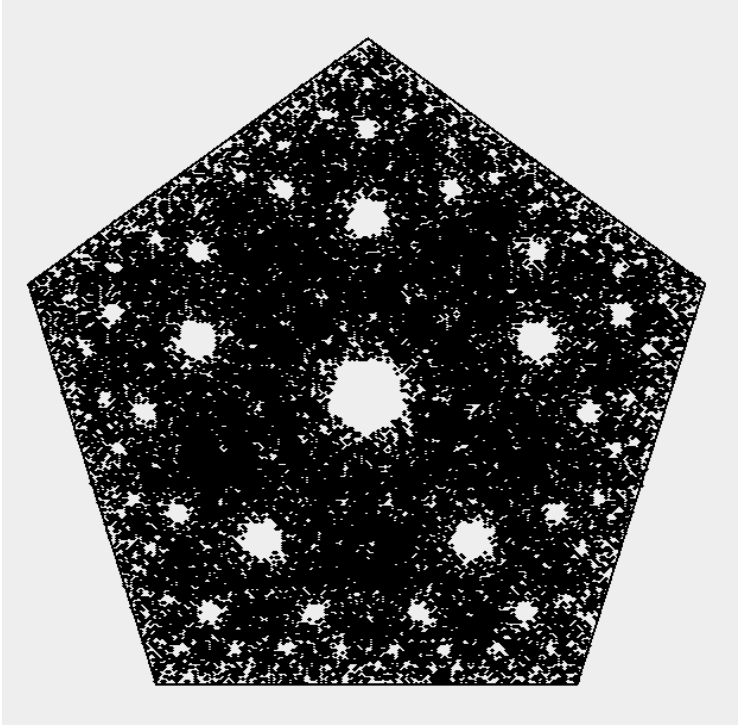
Exercise 3

Exercise 3 demonstrates the process of creating a self-similar graphic by iteratively using an algorithm that combines randomness and preset starting points. This program demonstrates how randomness and predetermined rules set by the algorithm can create complex and interesting patterns.



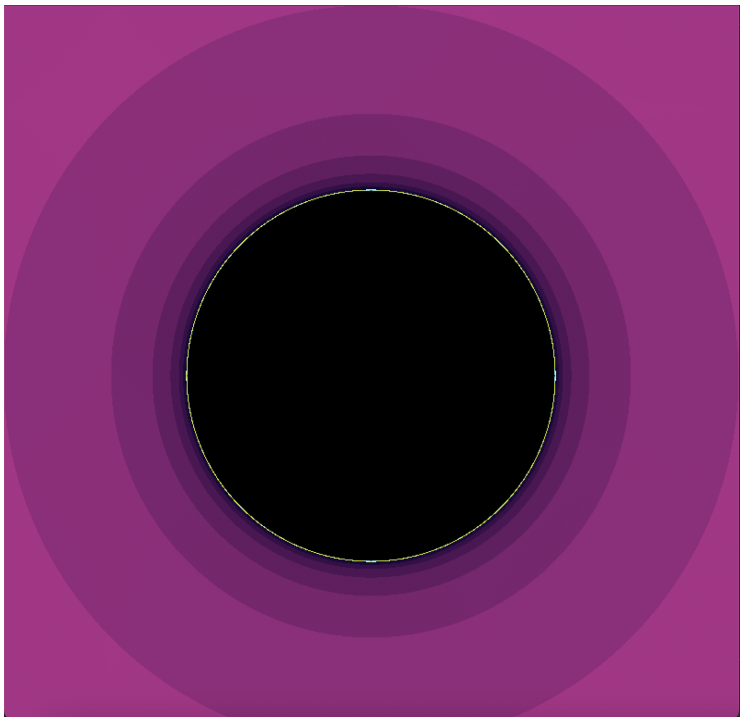
Exercise 4

Exercise 4 is just like exercise 3 however, instead of providing a specified particular algorithm, this task allows for artistic interpretation and encouraged us to design our own algorithm to generate an intriguing and visually appealing self-similar figure.



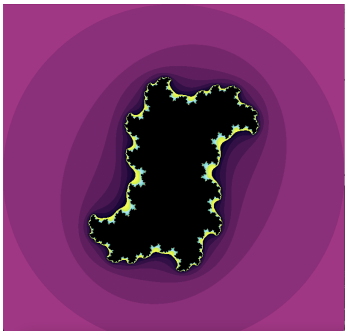
Exercise 5

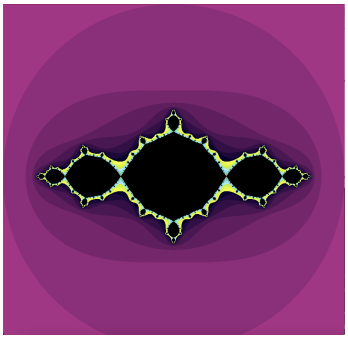
The purpose of this exercise is to generate a visual representation of the Julia set for the function w = z^2. The Julia set is a mathematical set that can be used to create complex patterns. The program follows a series of steps to determine whether each point in a specified region of the complex plane belongs to the "escape set" or the "prisoner set." The escape set is colored based on how quickly the iterations of the function approach infinity, while the prisoner set is colored black. This exercise took the skills learned in the past four exercises and bumped them up a notch and was very challenging but also very informative and I (Connor) learned a lot from this program.

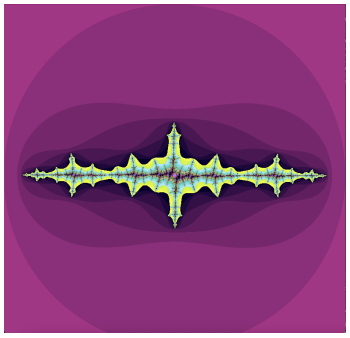


Exercise 6

Exercise 6 was not very challenging however it did provide a broader idea of what the program we created in exercise 5 is capable of.

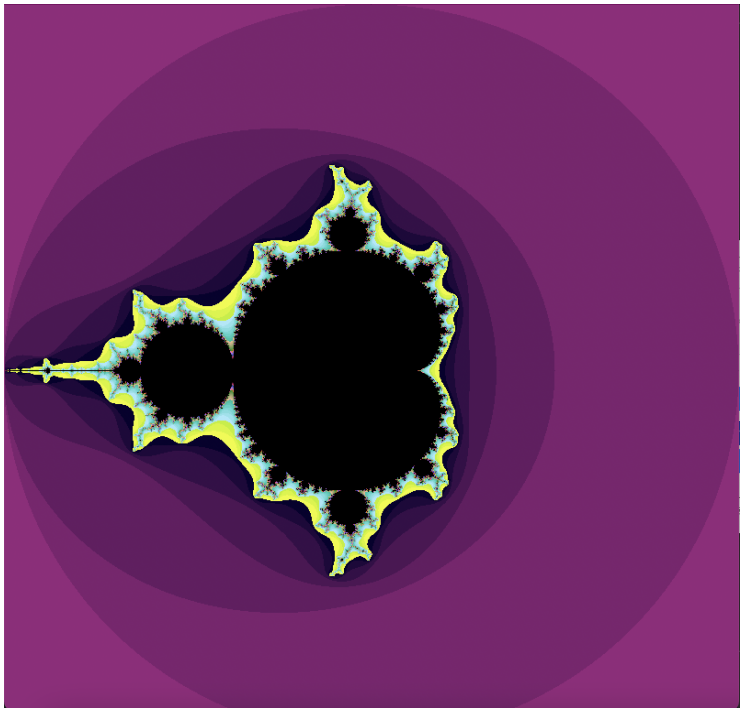


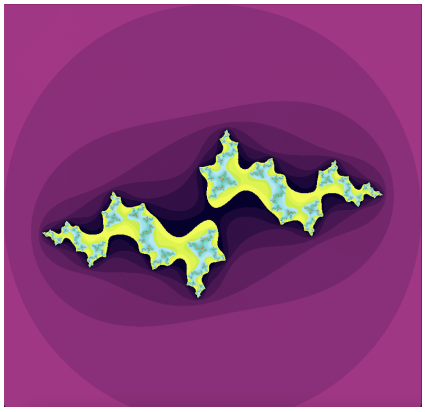


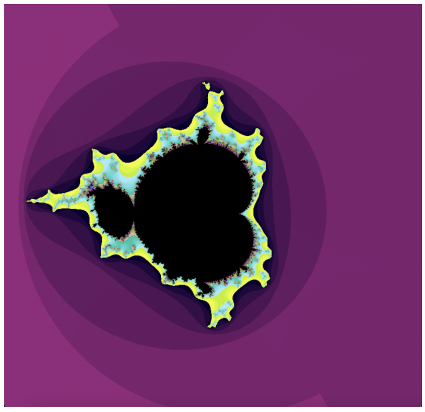


Exercise 7

Exercise 7 involves the implementation of the Mandelbrot set; a mathematical set of complex numbers generated through iterations of the function f(z) = z^2 + c. The program draws the Mandelbrot set by iterating over various complex constants, marking points inside the set as black and using colors to represent other points (Similarly to the Julia set). This allows for the creation of some pretty awesome images.







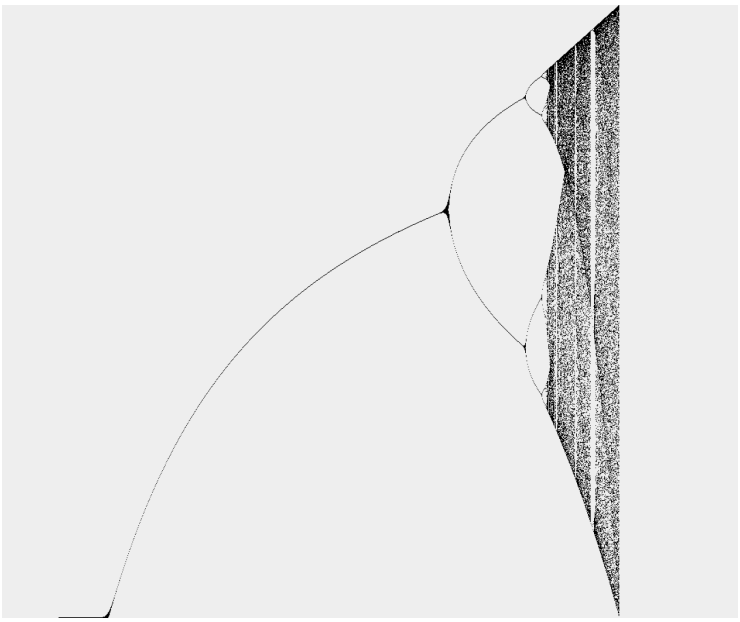
Exercise 8

Exercise 8 involves the exploration into the behavior of a population over time using a logistic equation (*y=rx(1-x)*). The goal was to find the approximate value of the growth rate *r* (*0<r<5)* at which the population transitions from a steady behavior to a more chaotic one. The equation is iteratively solved for various *r* values while also keeping the initial value for *x* at a constant of 0.5. The program will then print out about 20 values of *x* for each *r* to show the point where the population becomes chaotic.

```
r = 3.57, x = 0.8925
r = 3.57, x = 0.3425191875000001
r = 3.57, x = 0.8039632634887933
r = 3.57, x = 0.5626546139837953
r = 3.57, x = 0.8784856056671535
r = 3.57, x = 0.38109266730088165
r = 3.57, x = 0.8420239350424608
r = 3.57, x = 0.47488007145331196
r = 3.57, x = 0.8902472914076192
r = 3.57, x = 0.3488141740299899
r = 3.57, x = 0.8108999603119839
r = 3.57, x = 0.5474281364004845
r = 3.57, x = 0.88446954160295
r = 3.57, x = 0.3647939225392345
r = 3.57, x = 0.8272379603250942
r = 3.57, x = 0.5102076828405112
```

Exercise 9

Exercise 9 involves the exploration of the logistic equation *y = rx(1-x)* for different values of *r* (*0 <= r <= 4*) to investigate the behavior of population values over time. The program iteratively calculates and then plots the values of *x* on the y-axis against the corresponding *r* values on the x-axis. The goal is to visually deduce the point at which the system transitions from stable to chaotic behavior.



Exercise 10

Exercise 10 aims to demonstrate the sensitivity of the logistic equation, $f(x) = rx(1-x)$, to initial conditions for certain values of the growth rate r. The exercise involves selecting a specific r value and two initial x values that are very close to each other. The goal is to illustrate that the subsequent iterations for these two initial values diverge significantly, highlighting the system's sensitivity to initial conditions.

```
Iterations:
Iteration 1: x1 = 0.975, x2 = 0.97461
Iteration 2: x1 = 0.09506250000000008, x2 = 0.09650685681000008
Iteration 3: x1 = 0.33549992226562525, x2 = 0.3400538052547515
Iteration 4: x1 = 0.8694649252590003, x2 = 0.8752271376674086
Iteration 5: x1 = 0.44263310911310905, x2 = 0.4258979211159021
Iteration 6: x1 = 0.962165255336889, x2 = 0.9535846394297142
Iteration 7: x1 = 0.1419727793616139, x2 = 0.17261780200632312
Iteration 8: x1 = 0.4750843861996143, x2 = 0.5570014961036329
Iteration 9: x1 = 0.9725789275369049, x2 = 0.9623282348235952
Iteration 10: x1 = 0.1040097132674683, x2 = 0.14138515281110461
Iteration 11: x1 = 0.36344760197260056, x2 = 0.47344202636517235
Iteration 12: x1 = 0.90227842611257, x2 = 0.9722492287420059
Iteration 13: x1 = 0.34387106474913476, x2 = 0.1052245972150642
Iteration 14: x1 = 0.8799326467519814, x2 = 0.36719428728836767
Iteration 15: x1 = 0.4120396173349332, x2 = 0.906214306417506
```

Impressions

Overall this lab was very educational and we both learned a lot about finding our way around these problems and translating them into code that we can run and visualize using graphics. We both really liked doing these kinds of assignments because we felt that they had a big payoff when we were done because we got to see the awesome graphics once we were done. We have definitely learned a lot about implementing algorithms and finding little work arounds to make them work well in code. We also learned a lot more about making graphics within java because it is not something we touched too heavily upon in previous courses.