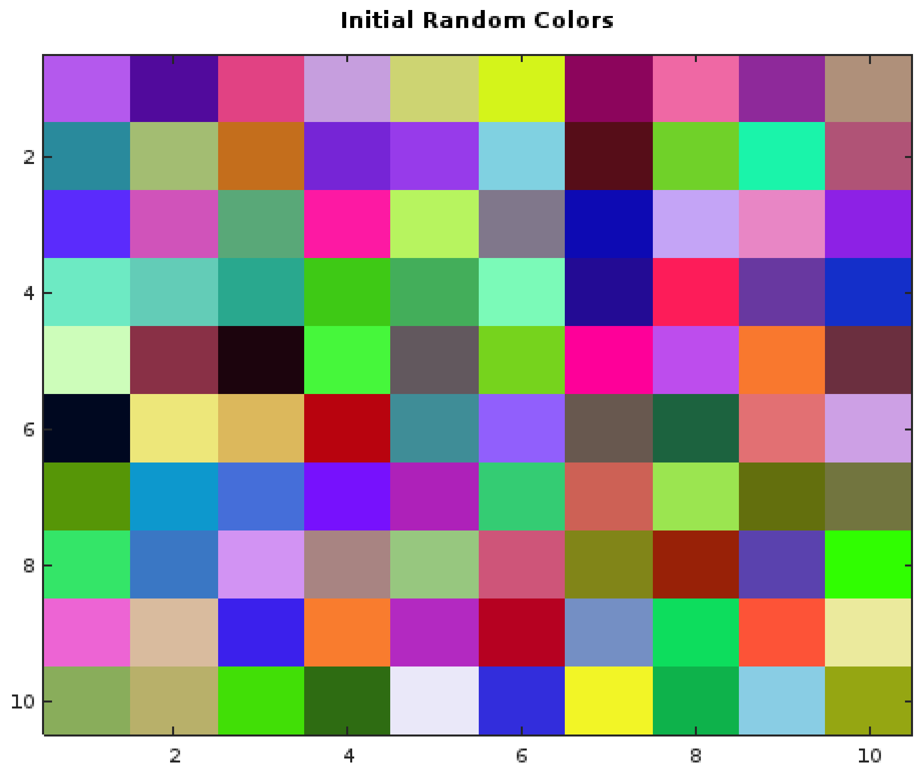
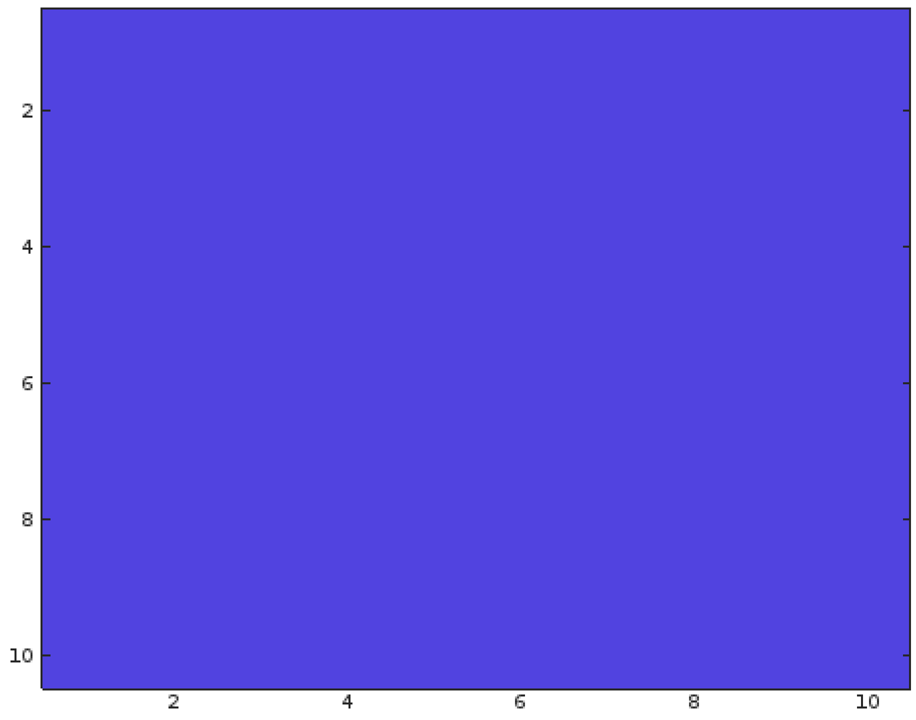
Jonathan Shipley Final Project

The purpose of this program is a sort of sorting method. Really, it's a way to make colors go head to head. The program generates a square of random pixels. They each have a color. It then uses simulated annealing to make those colors sort through based on their color.



That is how to looks to start. It is much more interesting to actually watch then it is to just see pictures. A possible final image can be



```
%*****************************************************************************************%
%*************** Final Project - Simulated Annealing Image Processing *******************%
% Jonathan Shipley
% Scientific Modeling
% 4/28/17
% Description: This program creates an image of random static. Each "pixel" will have a
%                        RGB color value that ranges from 0 to 1. So many colors are possible. This
%                        program will then sort that random image using simulated annealing.
%                        Each pixel will have a "heat" value associated with its color, the brighter the color,
%                        the higher the head value. When two pixels are next to each other, there is a chance
%                        that the pixel changes its neighbor's color based on its 'heat' value and the overall
%                        'energy' of the system. When temperature is high, there is a greater chance that pixels
%                        will act outside of the usual switch cases. This prevents 'hot' colors from winning
%                        right from the start. Let's see which colors wins.
%*****************************************************************************************%
%*****************************************************************************************%

% clear previous runs
clear;

Msize = 10;
M = rand(Msize, Msize, 3);
image(M);
title('Starting Randomizaiton');
% corresponding starting temp value corresponds to the color of the randomized image.
InitialCost = getEnergy(Msize, M)
cost = InitialCost;
temperatureInitial = InitialCost;
temp = temperatureInitial;

while temp > 0.5 * temperatureInitial
        for k = 1:100
                for a = 1:Msize
                        for b=1:Msize
                                % swap two neighboring pixels
                                randX = randi(0:1);            % generates rand 0 or 1
                                randY = randi(0:1);
                                c = a + randX;
                                d = b + randY;
                                if c <= Msize && d <= Msize
                                        MNew = M;
                                        MNew(a, b, :) = MNew(c, d, :);
                                        MNew(c, d, :) = MNew(a, b, :);
                                        costNew = getEnergy(Msize, MNew);
                                        dCost = costNew - cost;
                                        if dCost < 0 || exp(-dCost/temp) > rand
                                                M = MNew;
                                                cost = costNew;
                                        end
                                        image(M);
                                        drawnow();
                                end
                        end
                end
        end
        temp *= 0.90 * temp;
```

```
	cost = getEnergy(Msize, M);
end
% returns the average "energy" of an image
% energy will be defined as the "noise" of the image

% for example, the energy of two pixels will be the sum of the absolute value
% of the difference between the pixels' colors.

% then find the average for the image

function energy = getEnergy(Msize, M)

totalE = 0;

for a=1:Msize
	for b=1:Msize
		if b ~=Msize && a~=Msize
			energyR = abs(M(a,b,:)-M(a+1,b,:));
			totalE += energyR;
		end
		if a~=1
			energyU = abs(M(a,b,:) - M(a-1, b, :));
			totalE += energyU;
		end
	end
end

energy = sqrt(sum(totalE.*totalE));
```