# COMP 171 ‐ Lab 5 – Broken Screensaver

**NAME:**  Kristina Keenan

1. Watching the screensaver work, what is your initial idea for an algorithm for making a screen saver like this using turtle?

Turtle and random are imported, then a turtle is created. That turtle is assigned a random color and then is given a random starting position. The turtle then is told to move forward a certain number of spaces, creating a line, picks it tail up, then is returned to the start of that line. The turtle is moved x degrees to the left or right (a very small number), and its color is changed by y (very small number). The turtle then continues drawing these lines, whose lengths increase or decrease until a certain number of lines are reached, until it hits the edge of the screen, and then it reverses its direction to avoid falling off the screen.

1. After examining and cleaning up the code presented, what is their algorithm (and how was it different from yours)?

Their algorithm is to first call the main function, which then calls the function drawLines. Then a width, height, and four starting coordinates (before and after x’s and y’s) are defined. The function coordinateChange is then called to set the increments that each of the coordinates will change by. The function coordinateChange selects two random numbers and uses those numbers to determine a positive or negative number to set the coordinate increments by. Then the program selects six random numbers, and uses those numbers to determine the increments by which the color values (RGB) will increase or decrease by. Then, the program determines the color, interval, and coordinates and then creates a turtle and draws a line, changing the color and direction. If the number of lines equals or is over 100, or the color variables exceed or equal one or are less than or equal to 0, all of the variables determining the characteristics of the lines are changed. The program also prevents the lines from exceeding the bounds of the width and height. This algorithm is different from my algorithm in that the actual algorithm relies more on if statements and more random numbers are generated.

There were **two main problems** in the code you were given.

1. What was the first problem?

The incrementing RGB values for the lines would end up having one or more value go over 1 or below 0, causing the program to crash.

* 1. Was the error a **syntax**, **runtime**, or **semantic** error?

(see chapter 1 "debugging" section if you don't remember these terms)

Runtime Error

* 1. How did you fix the problem?

I added to the if statement “if a >= 1000:”, making it initiate when a>=1000 or when either of the RGB values (s1, s2, s3) is greater than or equal to 1 or less than or equal to 0, therefore changing the values when they meet or exceed the allowed values.

1. What was the second problem?

The drawing would either start as or after only a few turns continue in one direction, not turning like it is supposed to.

* 1. Was the error a **syntax**, **runtime**, or **semantic** error?

Semantic error

* 1. How did you fix the problem?

Under the if statement “if a >= 1000 or s1 >= 1 or s1 <= 0 or s2 >= 1 or s2 <= 0 or s3 >= 1 or s3 <= 0:”, I added these lines of code:

finc=coordinateChange()

jinc=coordinateChange()

kinc=coordinateChange()

minc=coordinateChange()

These lines of code changed the increments by which the drawing goes by, ensuring that when one of the above conditions is met (a>=1000 or s1>=1, etc), the drawing would change increments for each coordinate instead of going in one direction.

I also added the conditions f<0, j<0, k<0, m<0 to:

f = f + finc

if f>width or f<0:

finc = finc \* -1

j = j + jinc

if j>height or j<0:

jinc = jinc \* -1

k = k + kinc

if k>width or k<0:

kinc = kinc \* -1

m = m + minc

if m >height or m<0:

minc = minc \* -1

This causes the program to bounce the drawing off of the bounds of the window.