I started this project for fun, three days ago. Read a lot about the intrinsic operations hidden in the pattern. I’ll try to describe them in simple words. The algorithm is almost similar to the Ulam Spiral code. Here I need to operate on every *(x,y)* point in the Cartesian plane that is generated by the functions, but both the abscissa (*x*) and the ordinate (*y*) will have to be separate recursive functions.

* We input a length size *l* and number of iterations *n*.
* We make a function named dragoncurve and two recursive sub functions *x(n)* and *y(n)*.
* Run a for loop
* Whenever *n* is zero, we use a return statement that stops the program, other than that...
* Refer to the diagrams below, you’ll understand what follows a little better...
* First we use the *x(n)* recursive function. Move horizontally backwards, rotate right by 90, only changes *x* by one unit, this is resembled by x(n−1)x(n−1)
* Move down by one unit, resembled by *y(n−1)* and move forward by a length *l*.
* Now we move on to the *y(n)* function. If *n=0*, we stop the program, otherwise
* Move forward. Now here’s a trick, you’ll always have to keep track of which way you’re going, that way your left and right turns, forward and backward traversal could be taken care of effectively, when we put turtle.forward and *x(n−1)* and *y(n−1)*
* Finally, we call the function dragoncurve and input a length and iteration size.

Now, be careful about what you put in as the input when you call the function. Too large will crash the computer. I put in a length of 55 units and 1515 iterations and left the program running for 2020 minutes. A 7th generation i7 and a 4GB GTX 960M was not good enough to provide me with a faster execution time, even after I set the drawing speed to 10K10K. The output was so big that I literally had zoom out to 5050 of the original screen ratio.