Thu Le's Project 10

Project 10: Non-Photorealistic Rendering

Summary:

The goal of this project is to enhance the scenes that we can draw with our shape classes by adding new drawing styles and using parameterized strings – strings that can hold information about the distance, angle and pen width for each line drawn. We started the project in lab by first modifying our TurtleInterpreter and Shape classes so that they can draw a line in either the "normal" style or the "jitter" style. I then wrote codes for three more styles: "jitter 3", "dotted" and "brush", and used them to make a non-photorealistic version of my indoor scene from Project 9. The second enhancement we did in lab was adding methods to our Lsystem and TurtleInterpreter classes so that it can import information about distance, angle and pen width from the strings to make the drawings more diverse. To make use of this new enhancement, I created a new L-system that draws a maze and includes features like varied distance and multiple stochastic rules. My main extensions include simulating the effect of gravity on the trees and using multiple characters in a complex goal-oriented design in my maze-drawing L-system.

The assignment:

Task 1 – creating new styles

The new styles I created are called "jitter 3", "dotted" and "brush". They are in the TurtleInterpreter class.

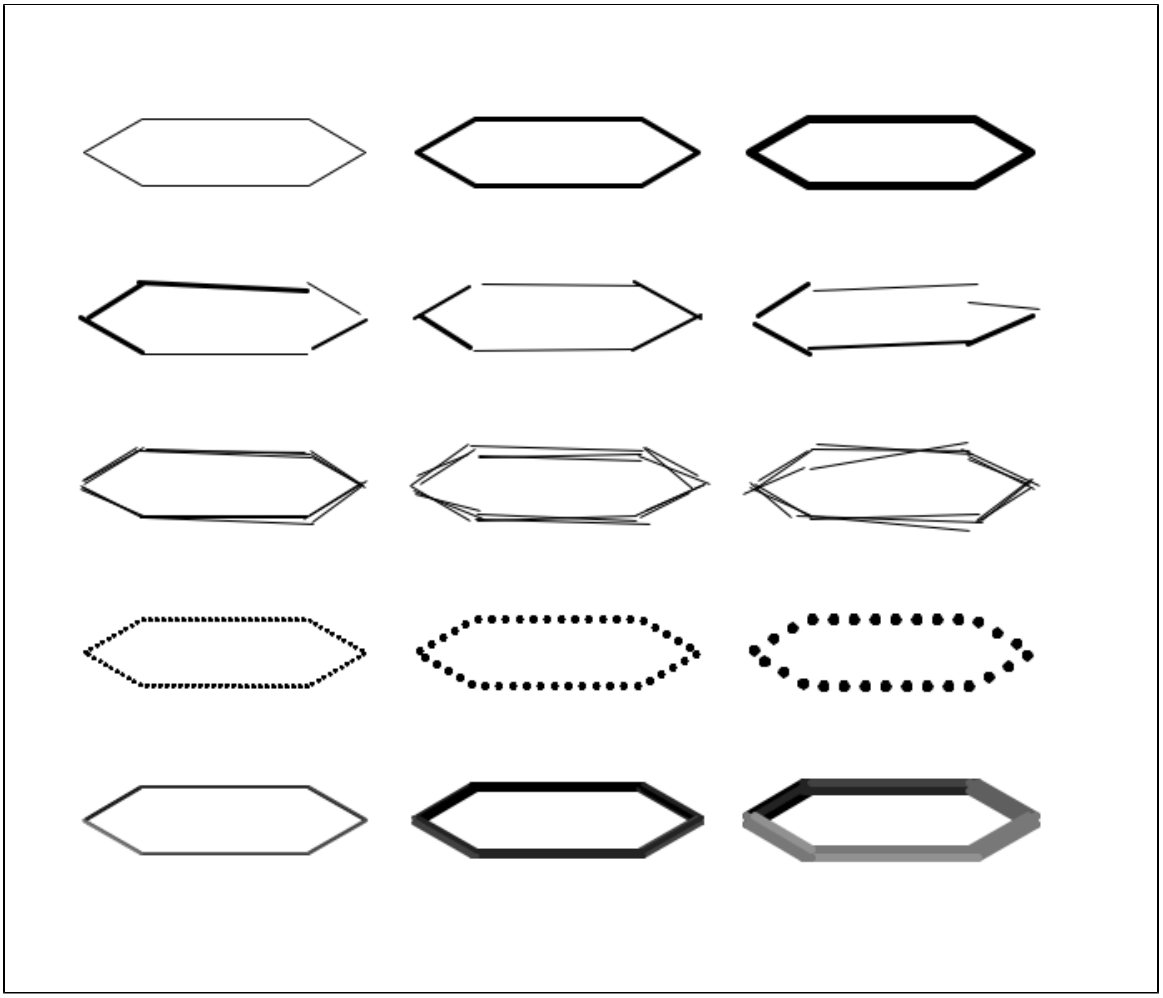
The code for "jitter3" has the same basic structure as the code for the "jitter" style we did in lab, except that "jitter3" draws three lines for each "F" in the string. I used a for loop to draw the three lines, which makes the code more concise. The start and end of each line differ slightly from the start and end specified by the string, and the values for this difference were generated from a Gaussian distribution using the random package.

For "dotted", I first calculated the number of dots to draw (N) based on the distance and dot size. I then used a for loop to draw N evenly-spaced dots in the same direction as the turtle was heading. To implement this style, I added a field called dotSize and a method to change the dot size to the TurtleInterpreter and the Shape classes.

For "brush", I drew four parallel lines with the same distance, but starting from four different locations at equal distance from the original starting point of the line. To draw these lines efficiently, I used a for loop and specified the starting point based on whether the loop control variable is even and less than 2. I also varied the color of the lines by specifying a 50% probability that the line has a lighter color than the original color:

```
self.color( (r, g, b) )
if random.random() < 0.5:
    self.color( (min(r + 0.1, 1), min(g + 0.1, 1), min(b + 0.1,1)) )
```

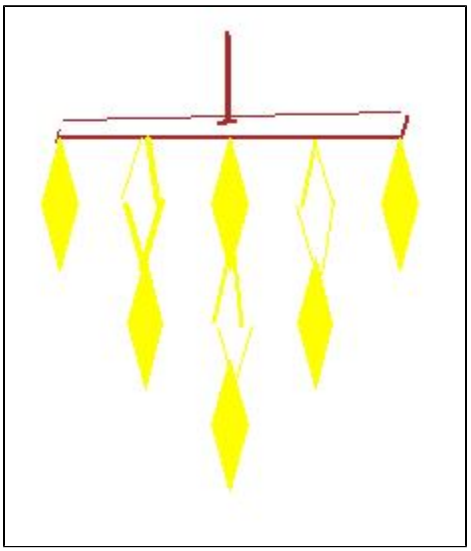
Below is a demonstration (in demo_line_styles.py) of my drawing styles using the Crystal shape class I created for Project 9. Each row represents a style, which is "normal", "jitter", "jitter3", "dotted" and "brush" from the top down. From left to right and depending on the style, the object has increasing width, jitter sigma or dot size.



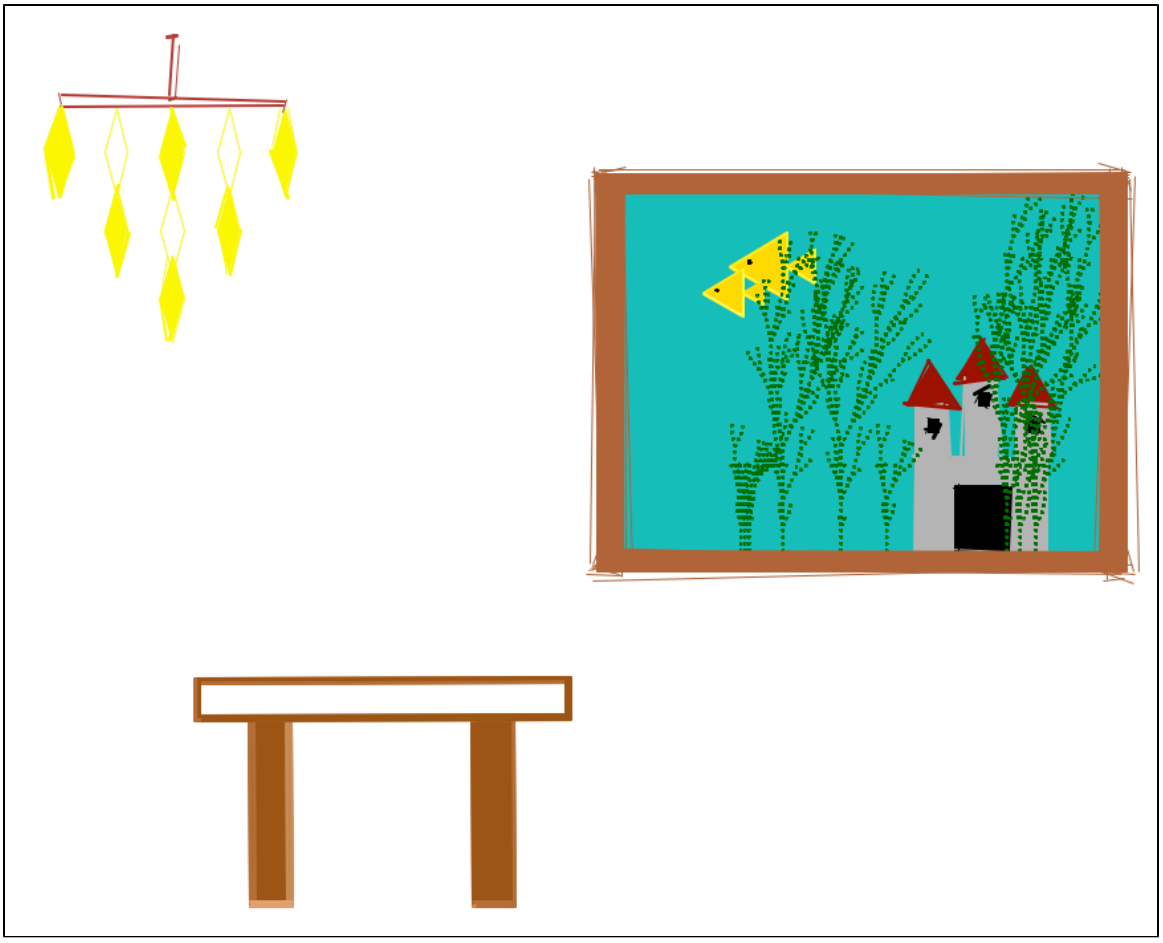
Task 2 – enhancing the indoor scene from Project 9

I assigned a style to each one of my objects: the table and the fish have the “brush” style, the chandelier and the castle have the “jitter” style, the tree has the “dotted” style and the picture frame has the “jitter3” style.

A problem I had was that when I tried to draw filled shapes, the shape always appears as if it has the “normal” style. In the example is shown below, the filled diamonds in the chandelier seem to have the “normal” style even though all the diamonds have the “jitter” style.

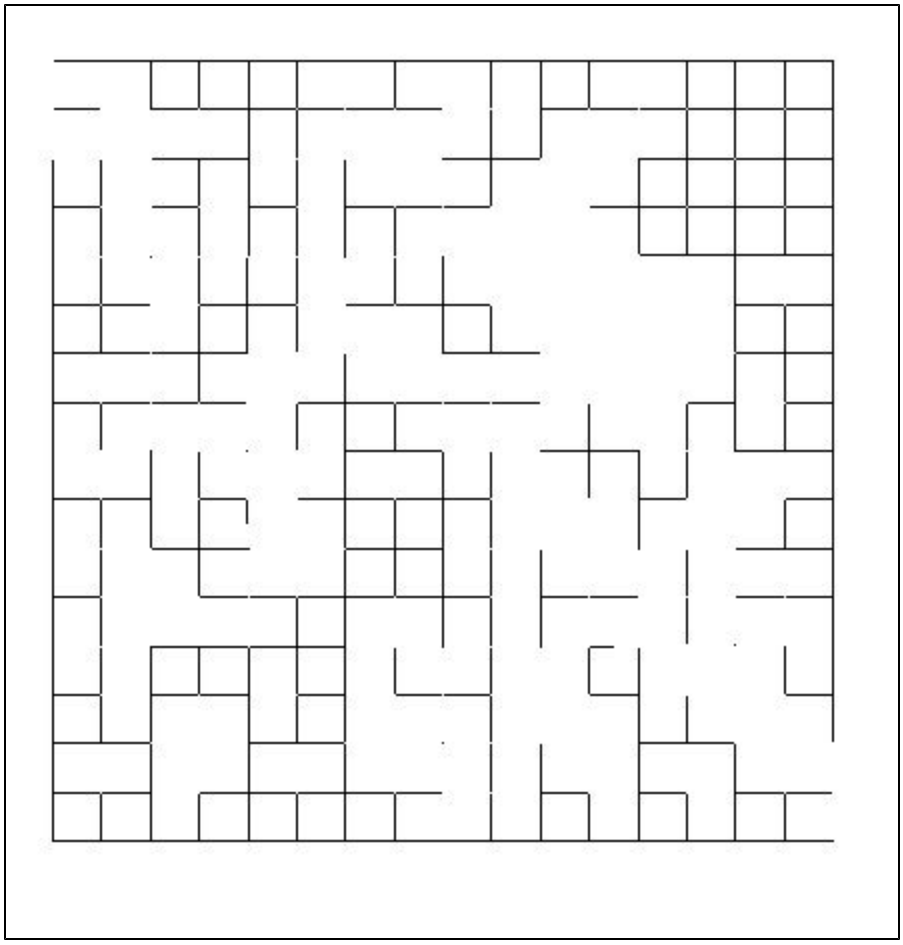


I fixed this problem by drawing each filled object twice: the first time with the filling and the second time with only the outline. Below is my enhanced indoor scene.



Task 3 – creating a new parameterized stochastic multi-rule L-system

I created an L-system to draw a square maze (square_maze.txt). An example output image with 4 iterations is shown below:



My base string is the outside wall of the maze, which is as follows:

```
(1)F-(1)F[<--w(0.125)f>]-(1)F-(1)F[<--w(0.125)f>]
```

The part [<--w(0.125)f>] tells the turtle to go back by a short distance and color that part white (“w” stands for the color white), which creates the entrance and exit of the maze. In addition, I use f instead of F because I want to create new branches from F, but not from the white space drawn by f.

My L-system has 4 rules. The first rule make some inner walls from the outer wall:

rule (x)F (0.5*x)F[-(0.5*x)A](0.5*x)F

A is another character in my L-system vocabulary that means “going forward” (identical to F). The reason I use another character is that I do not want to draw lines outside of the outer walls of the maze, so I need two separate rules for the outer walls and the inner walls. The first rule is thus for the outer walls.

The second rule is a stochastic rule for the inner walls (not show here since it is very long). This rule allows each (x)A character to branch to the left (25% chance), the right (25%) or both direction (50%) at the middle. I increased the percentage of the last choice by adding a copy of it to the rule. In addition, I added an at sign (@) to the beginning and end of each “forward” character, which will be used in the third rule.

The third rule add white spaces to the walls of the maze, which serve as doors connecting one section to another:

rule @ @ [<--w(0.125)a>] [<w(0.125)a>]

This means there is a 2/3 chance of inserting a white space at each @ symbol and 1/3 chance of leaving the at symbol intact. Half of that probability is for inserting a “forward” white space (w(0.125)a), which is useful at the beginning of a “forward” character, and the other half is for a “reverse” white space (--w(0.125)a), which is useful at the end of a “forward” character. The character “a” also means going forward (same as A and F) and is used for the same reason as f.

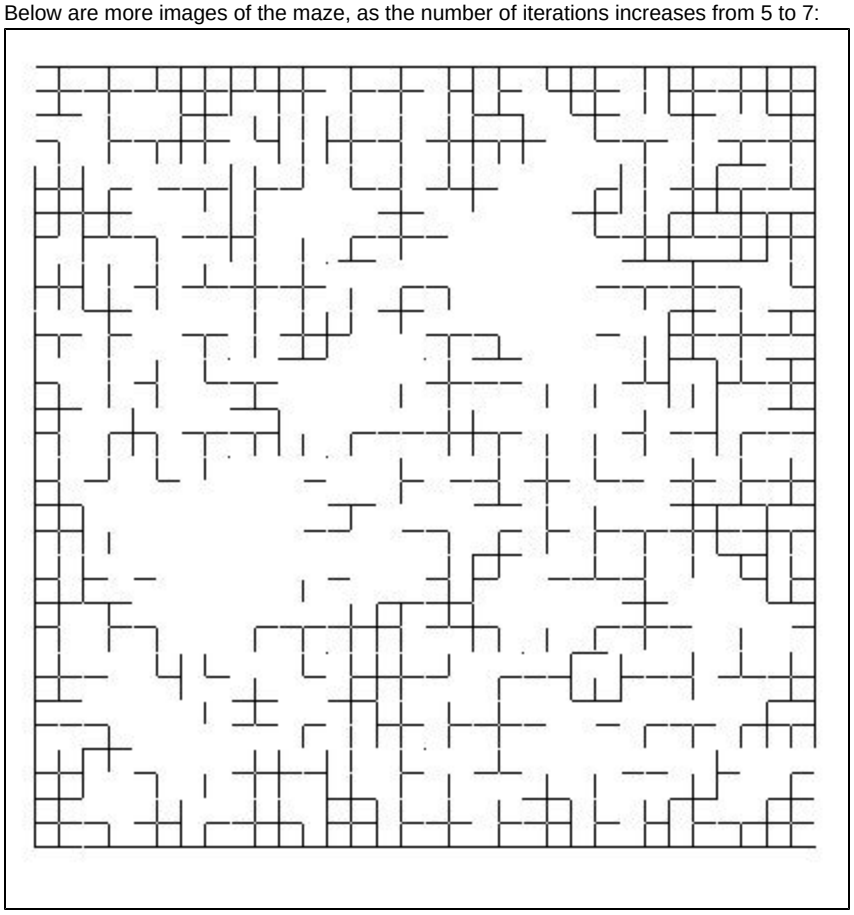
I ran the program after creating these three rules and noticed that there tend to be closed walls near the two exits, making the maze unrealistic. This is because the exits are at the corners, so the wall there were added to the string in the last iteration. Therefore, white spaces were not

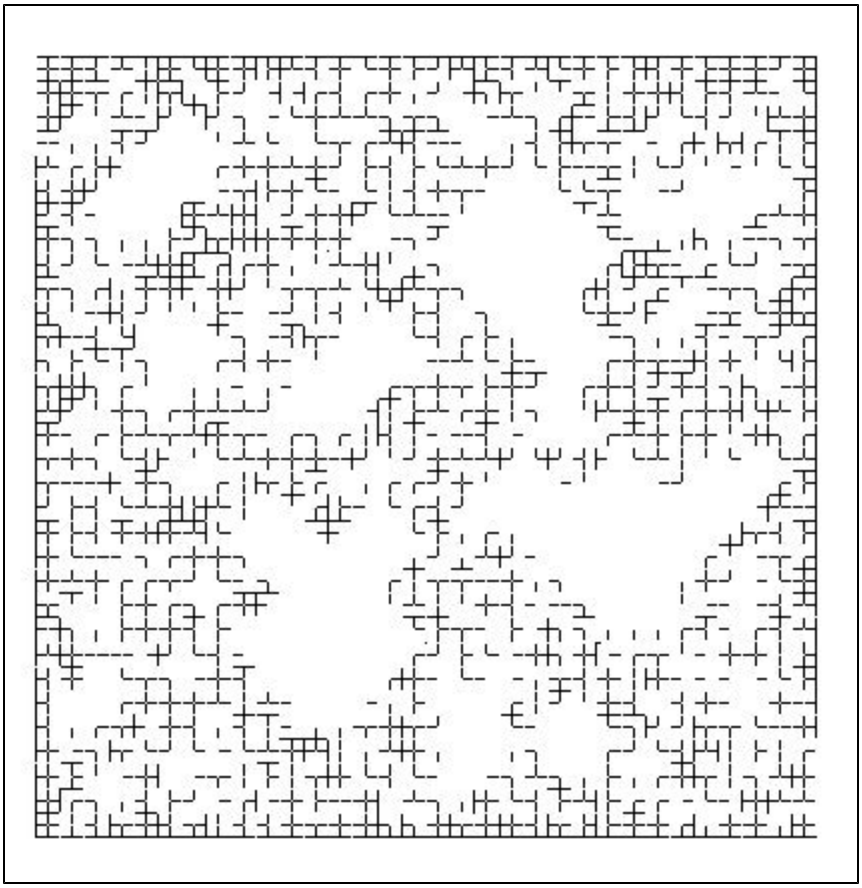
inserted into those areas. To fix this issue, I added a fourth rule:

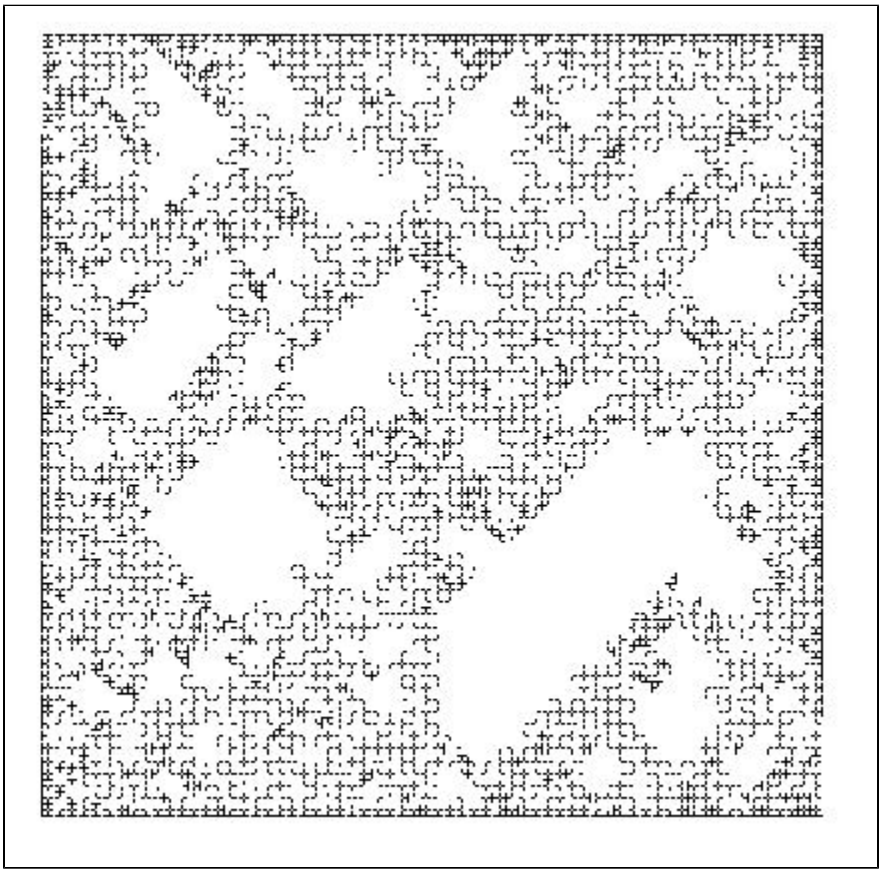
rule (x)f (x)f[(0.5*x)a(x)a] (x)f[(x)a(0.5*x)a] (x)f[(0.5*x)a(0.5*x)a] (x)f[(x)a(x)a]

This rule draws more white spaces at the entrance and exit of the maze. This is also the reason why I used two different characters – f and a – for white spaces, since f represents the entrance and exit while a stands for the doors inside the maze.

Below are more images of the maze, as the number of iterations increases from 5 to 7:





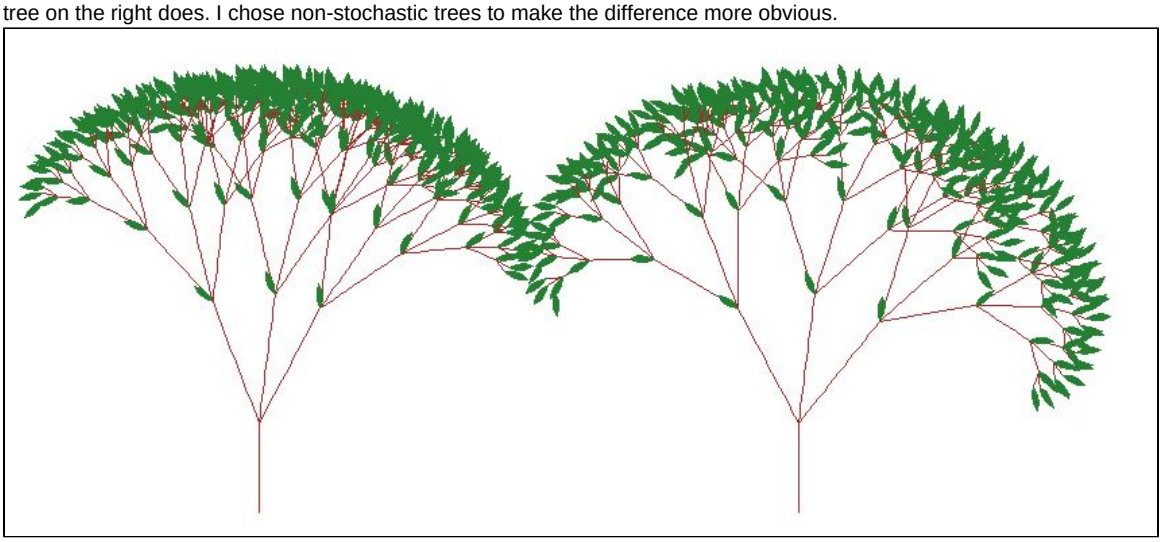


The L-system is stochastic, so the maze is different each time it is drawn. I noticed that when the iteration number is too small, there are not enough doors. An iteration number of 5 and above can produce a fairly realistic maze. When the iteration number is above 6, an interesting pattern was created that resembles a map or satellite image.

Extensions

Extension 1 – gravity:

I added the field “gravity” and the method setGravity to the TurtleInterpreter class and Shape class to store an update information about whether a tree should be drawn with gravity simulation or not. I then modified the drawString function in the TurtleInterpreter class such that when gravity mode is on, a left turn is increased by 50% when the turtle heading is in the second and third quadrant while a right turn is increased by 50% when the turtle heading is in the first and fourth quadrant. An example is shown below. The tree on the left does not have gravity effect and the tree on the right does. I chose non-stochastic trees to make the difference more obvious.



Extension 2 – maze-drawing system

My maze drawing system uses four characters: F, f , A and a for the same turtle process (going forward) so that I could differentiate between the outer and inner walls and between the two main exits and the inside doors. I also added another character, @, to efficiently add doors to the maze. In addition, I figured out how to increase the probability of a choice in a stochastic system, which is by including multiple copies of it. Finally, the design of my L-system is goal-oriented. Since I wanted a maze where the entrance and exit are preferably connected (but not necessarily), I added the fourth rule to draw more doors near the entrance and exit so that the maze was not obviously unsolvable.

Other extensions:

I created an extra style - “brush”. I also figured out how to draw filled shapes with the styles. In my code, I try to use loop as much as possible to increase efficiency and arrange objects in rows and columns in test functions (for example in demo_line_styles.py).

What I learned:

Firstly, I learned how to add new functionalities to an existing collection of interconnected classes, since it is important to make sure the same fields and methods are available in connected classes like Shape and TurtleInterpreter. I also learn how to create a new L-system based on a few criteria about what shape it should draw and how many different components it should have.

Acknowledgement:

I worked on my own for this project.