Unity Procedurally-Generated Trees

By Hung Bui

Introduction

Background About Me

I am a third year Computer Science & Computer Game Science student at UCI. I have a love for making games, and have a deep interest in the visual-aspect of game-making. Procedural generation has been something that's always perplexed me.

Implementation

To procedurally generate trees, I first researched approaches to do this. I figured that a recursive tree-like datastructure would be needed to create trees. After researching a bit more, I found that L-Systems is a common method used. After looking more into it, I found L-Systems fit for purpose, as they allowed the flexibility I wanted.

L-Systems: So, what are They?

According to Wikipedia, Lindenmayer Systems are

"a parallel rewriting system and a type of formal grammar. An L-system consists of an alphabet of symbols that can be used to make strings, a collection of production rules that expand each symbol into some larger string of symbols, an initial "axiom" string from which to begin construction, and a mechanism for translating the generated strings into geometric structures"

In other words, L-Systems are a set of rules to expand on a string, replacing characters in the string with a new string. When rules are iteratively applied to the string, the string expands and generates a larger string.

For example:

Rule 1: "A" → "BC"

Rule 2: "B" → "CA"

Iteration 0: "A"

Iteration 1: “BC”

Iteration 2: “CAC”

Iteration 3: “CBCC”

So, how is this helpful? These rules help procedurally generate strings from just a simple start template, replacing existing characters. In a sense, the iterative process of applying rules is similar to recursion, breaking down existing components into smaller ones, which matches how a lot of trees branch! Larger branches are broken down into smaller twigs, etc.

L-Systems: Interpreting Strings

I researched how to apply L-Systems to trees, and found a interpretation scheme that interpreted strings in a recursive way.

Symbol	Action	Meaning
+	Turn right	Rotates current direction to the right(x dir)
-	Turn left	Rotates current direction to the left(x dir)
&	Pitch down	Rotates current direction forward(z dir)
^	Pitch up	Rotate currentl direction backward(z dir)
S	Scale	Scale the branch down
F	Draw Branch	Draw a branch, and update the position
[	Save state	Save the current position, direction, scale and store onto a stack
]	Restore state	Restore the position, direction, scale on top of the stack

With this way of interpretation, a string such as

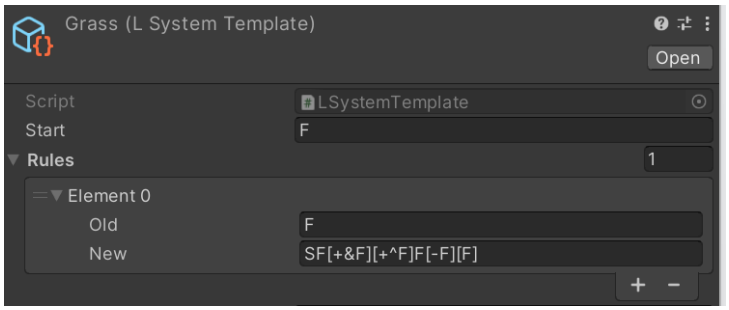
“F+[F]-[F]” would produce a tree in the shape of the letter “Y”

The first F draws a branch, then + rotates the direction to the right before saving the current state and drawing another branch in that direction. The state is restored, and another branch is drawn pointing left.

With the right LSystem rules, huge strings could be generated which when interpreted render as complex trees.

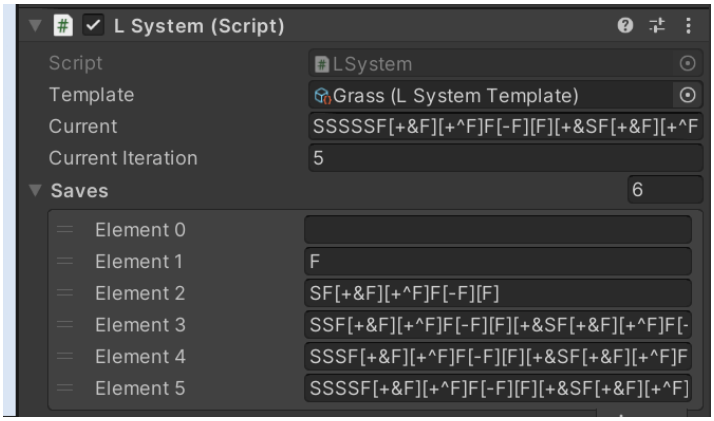
Implementation: Building Cusomizable L-Systems

Programming the L-System in C# Unity was not too difficult. It just consists of looping over a string and checking each character, and replacing it with a string. Here is an example of a set of rules:



Looking at the rules. Every iteration, every branch is replaced with a scaled version of that branch[SF], with two sub-branches[+&F][+^F] protruding in new directions, as well as another branch[F] with two more branches protruding[-F][F].

Here is the resulting string it creates after 5 iterations:



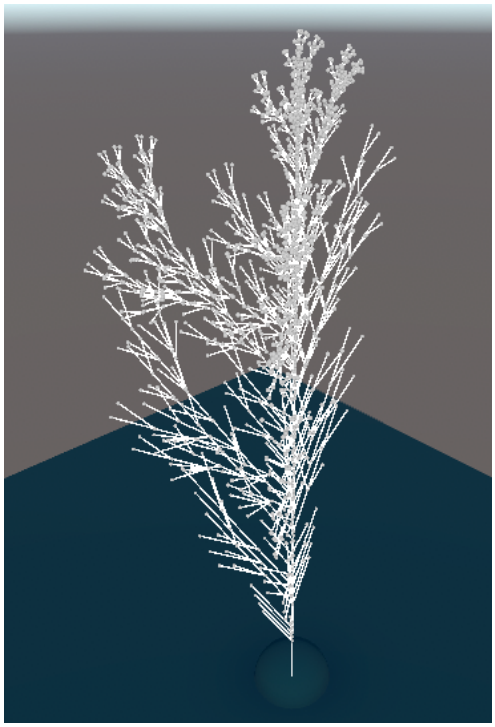
I created this ruleset by first viewing 2D templates online, and modifying them to make them 3D. Understanding how each symbol is interpreted helps a lot. The generated structure will be shown in the next section

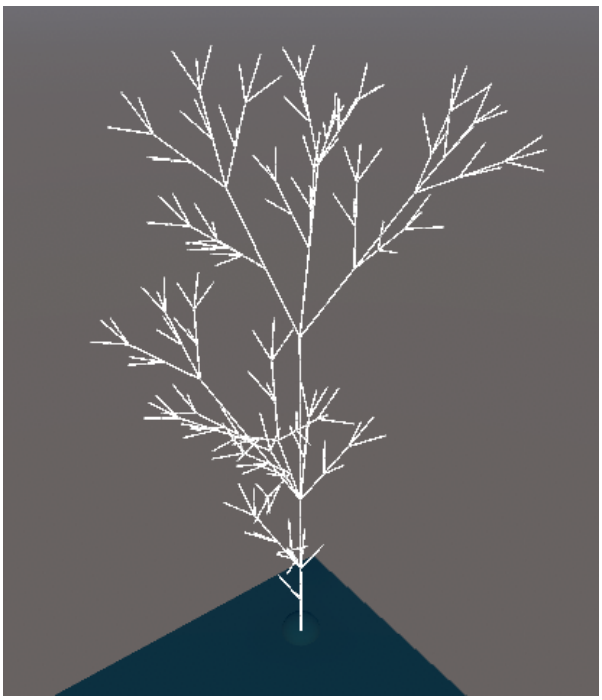
Implementation: String to Gizmo Rendering

Now that we are able to generate a string that represents the tree, it was time to interpret it. To do this, I made a parser, which holds the variables for position, branch direction, and current scale. I then mapped each of the 8 symbols to their own functions, which did different their corresponding actions. For now, the F symbol added a line which was drawn into the gizmo. Here are some results:

Gizmo Renderings

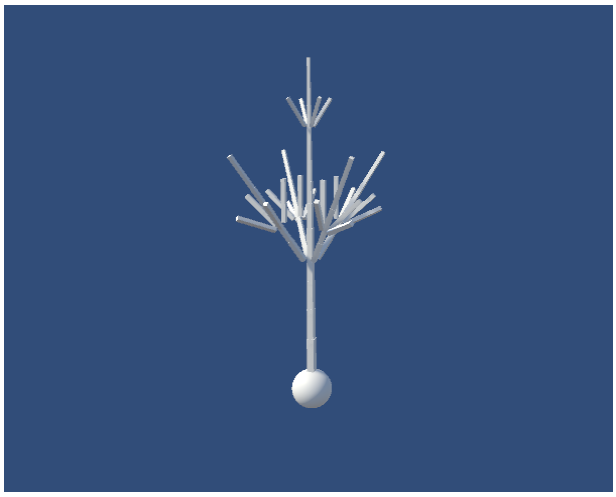
“Grass”, same L-System as prev section	“Fan Tree”





Implementation: Gizmos to Unity GameObjects

After the trees were correctly being generated in the gizmos, it was time to draw them in game view. This was more difficult than I thought. I took the approach of creating a branch prefab(a scaled rectangle) and instantiating them in the correct position, scaling them, and aligning the y axis with the current direction. This worked, and produced a result like this:



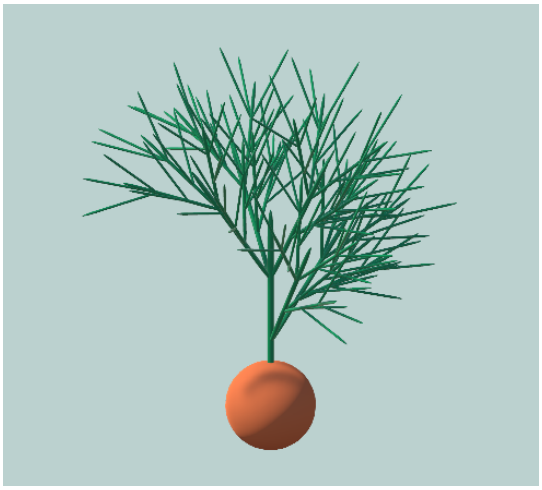
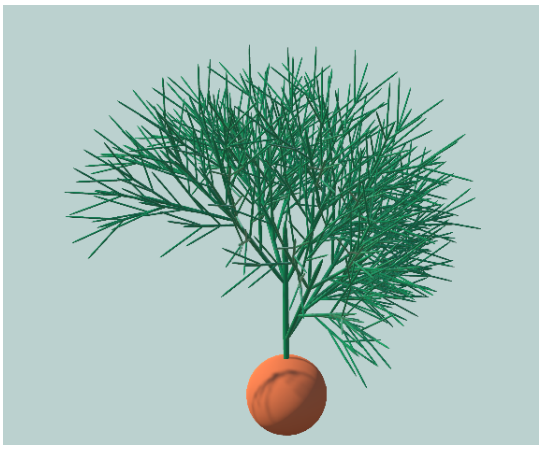
After more iteration and experimenting with branches, I found that scaling a sphere worked well in representing a stick.

Below is the result of the “Grass” mentioned in the previous sections, with slightly different settings on scale and angle.

Examples (pt 1)

Here are some more examples of different L-Systems rendered:

Tree Renderings with Different Iterations

Iterations	“Fluff Tree”	“Fan Tree”
4		
5	Forgot to screenshot	

6		

I found that after about 5 iterations, the amount of branches scaled so exponentially that performance was extremely hindered.

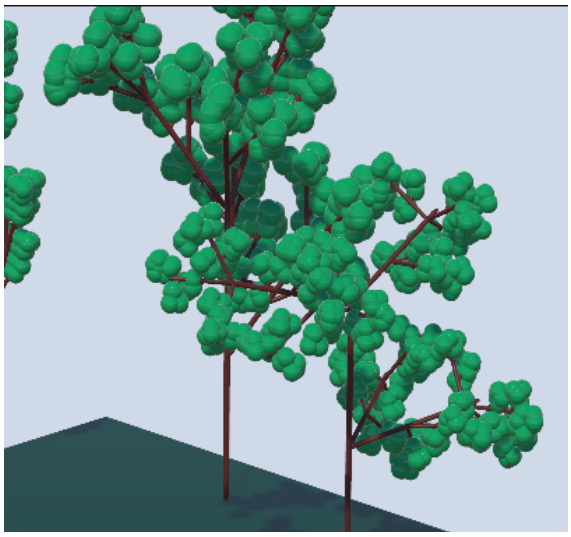
More Implementation

Adding in Leaves

I didn't want the trees to just be composed of branches, so I thought of ways I can add leaves. I considered replacing some branches with leaves, but this required I change the syntax/grammar of my L-System renderer, but I wanted to stick with using "F" to signify a branch.

After thinking for a while, I realized that leaves often grow at the end of branches. I realized that the symbol that corresponds to the end of a branch is "]", which restores the previous state. So, I modified that function to spawn a leaf gameobject at [the current position + 0.5 * the scale * the direction] before restoring the previous state. However, I limited this to only branches that had a recursive depth of greater than 2(I checked the length of the stack to get this value) This created results like this:

Leaf placement

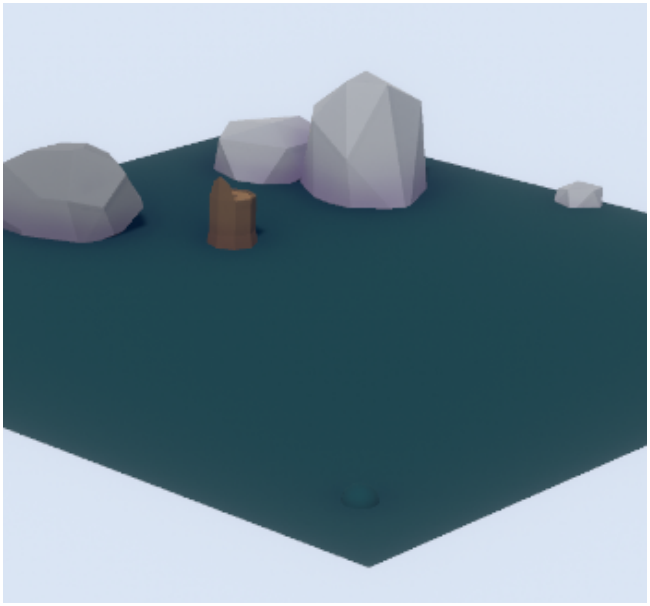
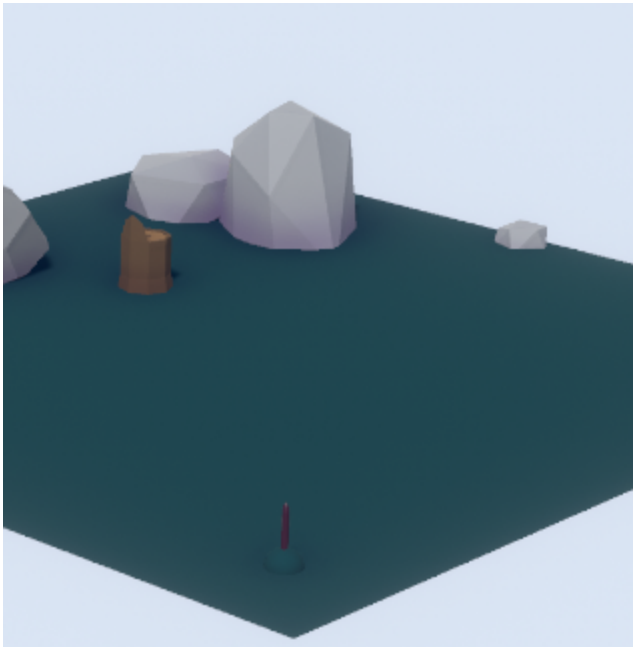


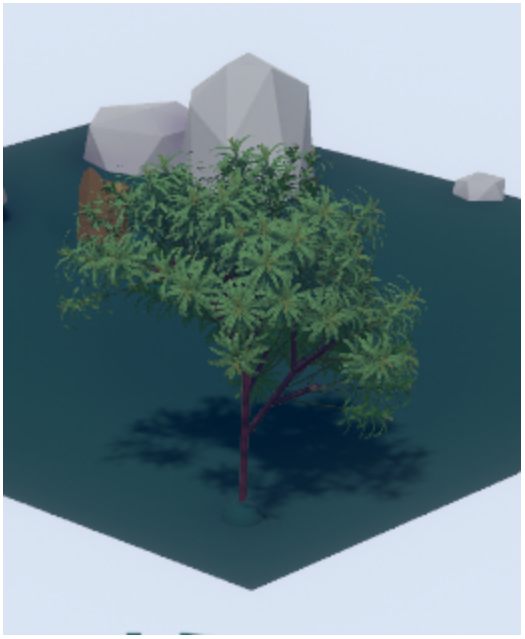
Animating Procedural Tree Generation

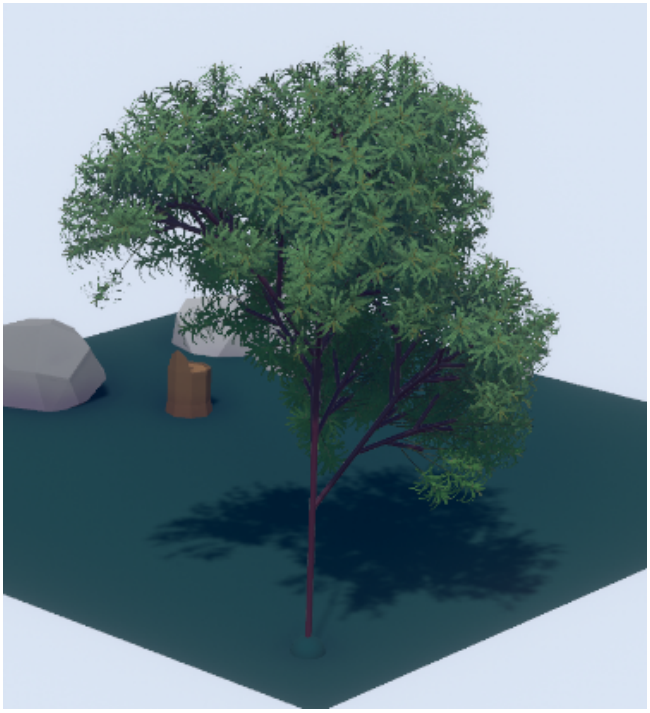
I also wanted some sort of animation to show the process of procedural generation. Well, since the L-system iterative works on a string anyway, I decided to store these strings into a list, and generate each tree for that list. Then, when I create a tree, I flip through these iterations from iteration 0 to iteration 5. This created the illusion that the tree was growing. Here is an example:

Animation Example

Iteration	Image

1	
2	

3	
4	

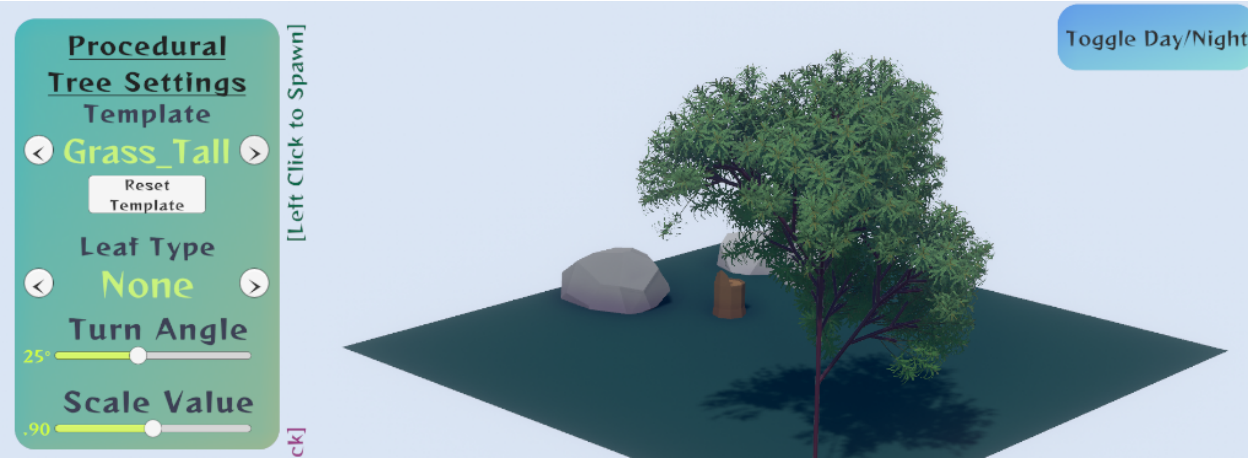
5	

Examples (pt 2)

Showcasing Customizations

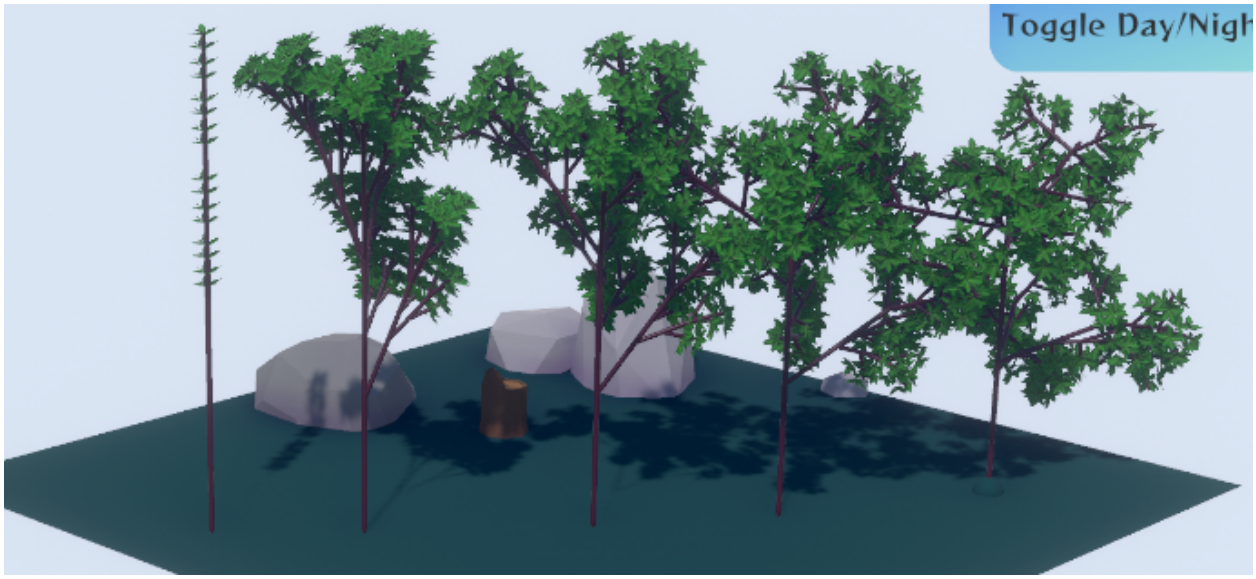
I wanted to allow the player to explore changing different parameters, so I created a UI that allows that. Players can test 6 L-Systems: “Grass”, “Grass_Tall”, “Fan Tree”, “Fluff Tree”, “Flowers”, “Bush”. I allow players to slightly alter the angle of rotation, and the scale between branches, as well as switch out what type of leaves to use! I imported different types of leaves from various different free asset packs.

Below is an example of the UI and customization menu:



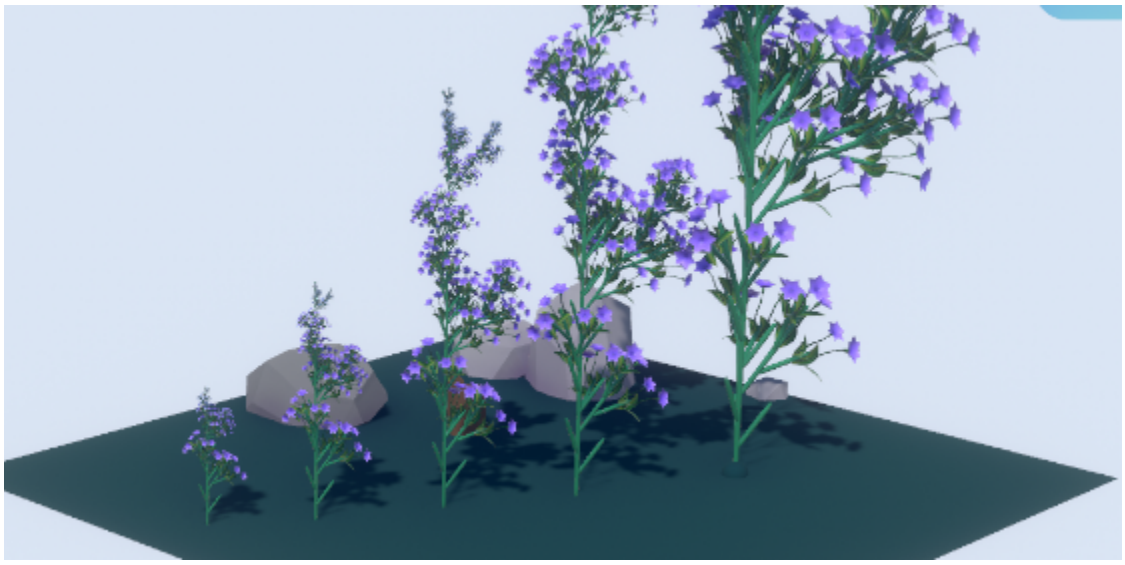
Fan Tree: Angle Variations

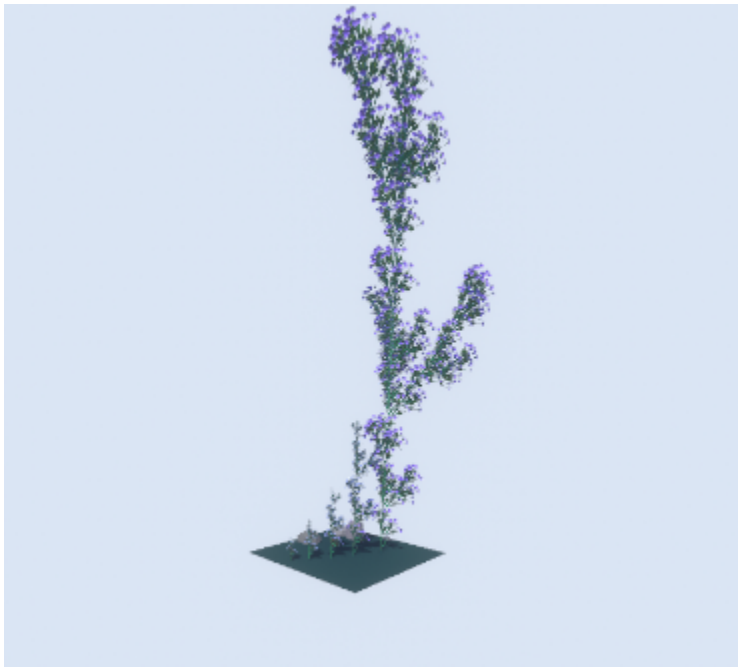
Here is an example of a “Fan Tree” with 5 different versions with Turn Angles ranging from 0 degrees to 60 degrees:



Flowers: Scale Variations

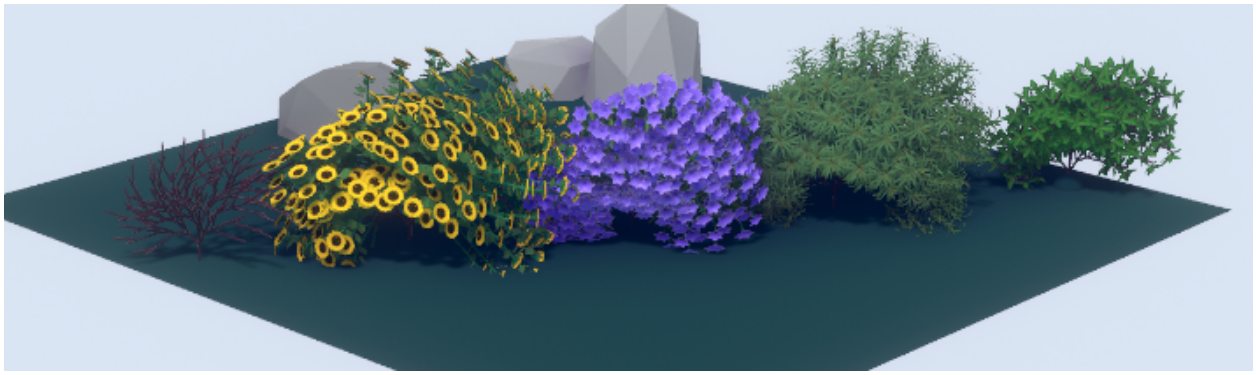
Here is an example of “Flowers”, with variations in scale value:

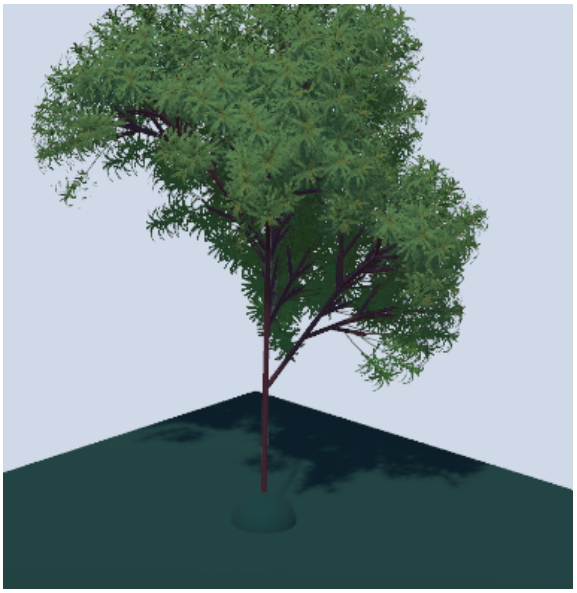




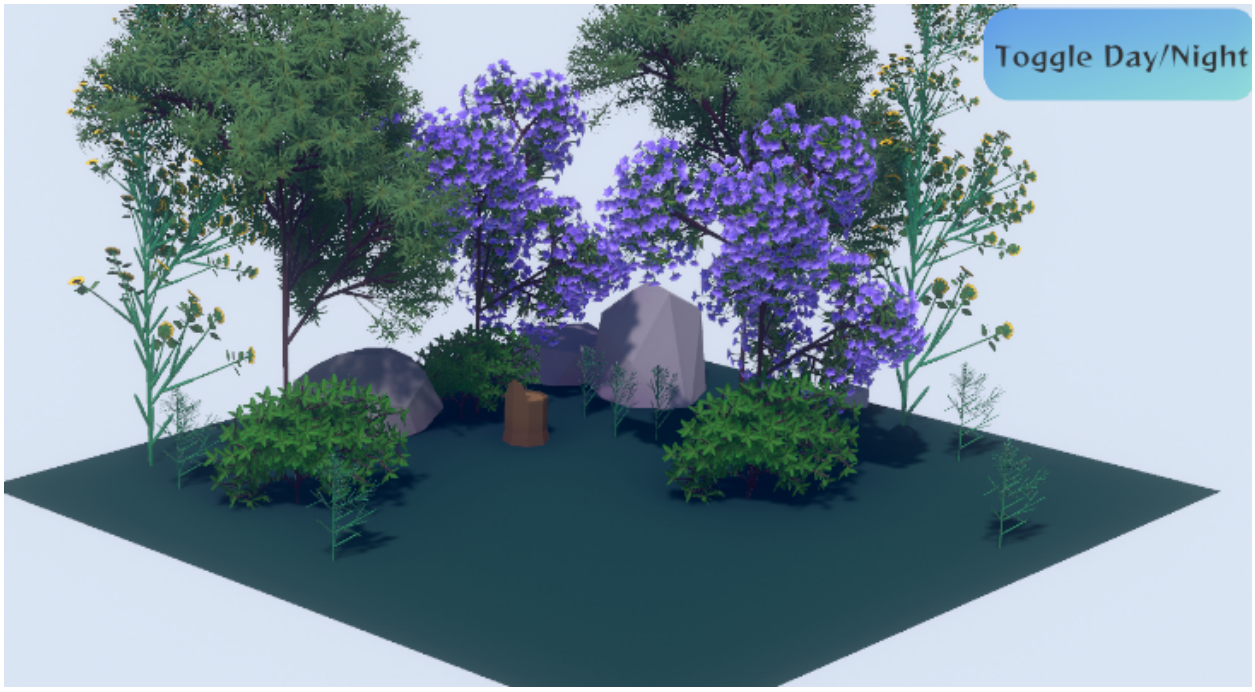
Bush: Leaf Variations

Here is an example of a “Bush”, with different leaf variations:





Everything put together



Conclusion

Possible Improvements

One possible improvement to this to make it more usable in actual games is to add some randomization between branch placement. This would allow for variation. Another thing I wanted to do that I couldn't get to is programmatically scaling each branch/leaf so that I could animate smoothly instead of showing frames of each iteration.

What I learned

I built a lot of implementation skills and refined my knowledge of data structures and Unity throughout this project. I was also able to really deepen my understanding of L systems and understand why it is such a common, efficient choice for procedural generation of trees. I was able to apply some problem-solving skills when it came to things I wanted to implement, such as animations and leaf placement, which I was able to come up with on my own.