```
using System.Collections;
using System.Collections.Generic;
using System.Text;
using UnityEngine;
using System;

public class TransformInfo
{
    public Vector3 position;
    public Quaternion rotation;
}


public class LSystem : MonoBehaviour
{
    [SerializeField] private int iterations = 4;
    [SerializeField] private GameObject Branch;
    [SerializeField] private GameObject Leaf;
    [SerializeField] private float length = 10f;
    [SerializeField] private float angleOfRotation = 45f;
    public float variance = 10f;
    private float[] randomRotationValues = new float[360];

    private const string axiom = "X"; //starting point for our tree,
      starting with this rule
    private string currentStr = string.Empty;

    private Stack<TransformInfo> transformStack;

    private Dictionary<char, string> rulesForL;


    void Start()
    {
        transformStack = new Stack<TransformInfo>();
        Debug.Log("Starting");

        for (int i = 0; i < randomRotationValues.Length; i++)
        {
            randomRotationValues[i] = UnityEngine.Random.Range(-1f, 1f);
        }

        rulesForL = new Dictionary<char, string>
        {
            {'X', "*=F+***=[[X*]-X]-=F[-FX]+X*" },
            {'F', "FF" }


        };

        Generate();
    }

```

```
53      private void Generate()
54      {
55          Debug.Log("Generating");
56          //starts as X, changes overtime
57          currentStr = axiom;
58
59          StringBuilder sb = new StringBuilder();
60
61          for (int i = 0; i < iterations; i++)
62          {
63              foreach (char c in currentStr)
64              {
65                  //adds value onto the string; if value is in the rules,
66                  //then we add value corresponding to key, else we add value
                      itself
67
68                  sb.Append(rulesForL.ContainsKey(c) ? rulesForL[c] :
                      c.ToString());
69              }
70              Debug.Log("Current String");
71              currentStr = sb.ToString();
72              sb = new StringBuilder();
73              Debug.Log("StringBuilder built");
74          }
75
76          //what we do for every char
77          for (int i = 0; i < currentStr.Length; i++)
78          {
79
80              Debug.Log("For loop starting");
81              switch (currentStr[i])
82              {
83                  case 'F':
84                      Debug.Log("Case F");
85                      GameObject fLine = currentStr[(i + 1) %
                        currentStr.Length] == 'X' ||
86                          currentStr[(i + 3) % currentStr.Length] == 'F' &&
87                          currentStr[(i + 4) & currentStr.Length] == 'X' ?
                        Instantiate(Leaf) : Instantiate(Branch);
88                      Vector3 initPos = transform.position;
89                      transform.Translate(Vector3.up * length);
90                      /*GameObject treeSegment = Instantiate(Branch);*/
91                      //0 - START POSITION
92                      //1 - next position to our transform position
93                      fLine.GetComponent<LineRenderer>().SetPosition(0,
                        initPos);
94                      fLine.GetComponent<LineRenderer>().SetPosition(1,
                        transform.position );
95
96
97                      break;
98
99                  case '*':
```

```
100                     Debug.Log("Case *");
101                     transform.Rotate(Vector3.up * angleOfRotation *
                    randomRotationValues[i % randomRotationValues.Length]);
102                     break;
103
104                 case '=':
105                     Debug.Log("Case =");
106                     transform.Rotate(Vector3.up * angleOfRotation *
                    randomRotationValues[i % randomRotationValues.Length]);
107                     break;
108                 case 'X':
109                     Debug.Log("Case X");
110                     break;
111                 //rotate tree clockwise
112                 case '+':
113                     Debug.Log("Case +");
114                     transform.Rotate(Vector3.back * angleOfRotation *
                    randomRotationValues[i% randomRotationValues.Length]);
115                     break;
116                 //rotate anti-clockwise
117                 case '-':
118                     Debug.Log("Case -");
119                     transform.Rotate(Vector3.forward * angleOfRotation *
                    randomRotationValues[i % randomRotationValues.Length]);
120                     break;
121                 //save current info
122                 case '[':
123                     Debug.Log("Case [");
124                     transformStack.Push(new TransformInfo()
125                     {
126                         position = transform.position,
127                         rotation = transform.rotation
128                     }) ;
129                         break;
130                 //return to previously saved
131                 case ']':
132                     Debug.Log("Case ]");
133                     TransformInfo ti = transformStack.Pop();
134                     transform.position = ti.position;
135                     transform.rotation = ti.rotation;
136                         break;
137                 default:
138                     throw new InvalidOperationException("Invalid Rule
                    Operation Thing");
139
140             }
141         }
142     }
143
144
145 }
146
```