LAB – 7

Gameplay, Game Mechanics, Game Balancing

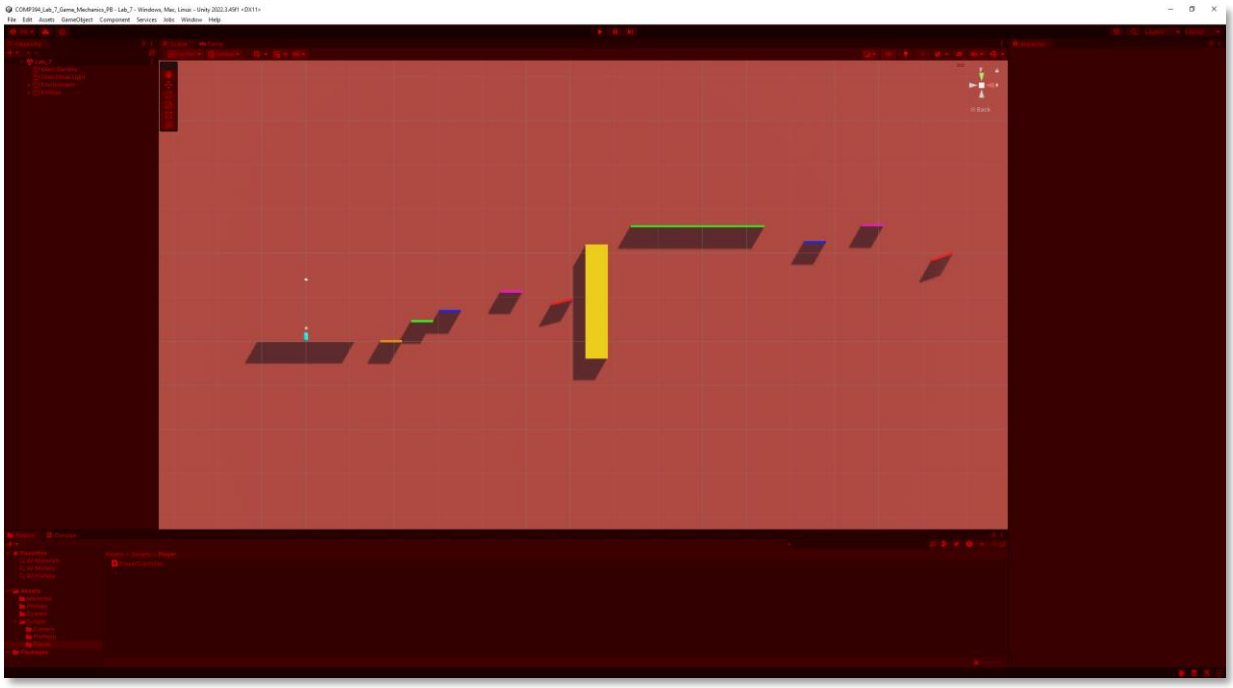


Fig 1.1: Level with platform with various forms of Transform modifications with sin (translate, scale and rotate)

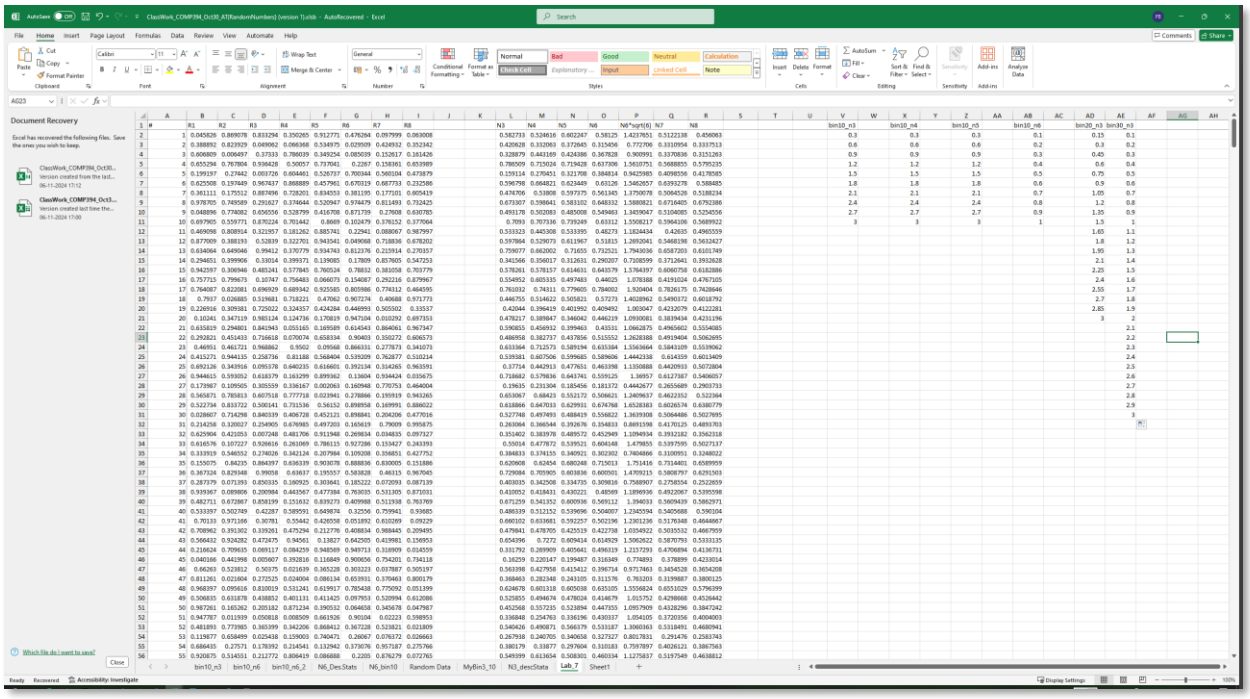


Fig 1.2: Added R7 and R8 along with N7 and N8. Also, added bin20_n3 and bin30_n3.

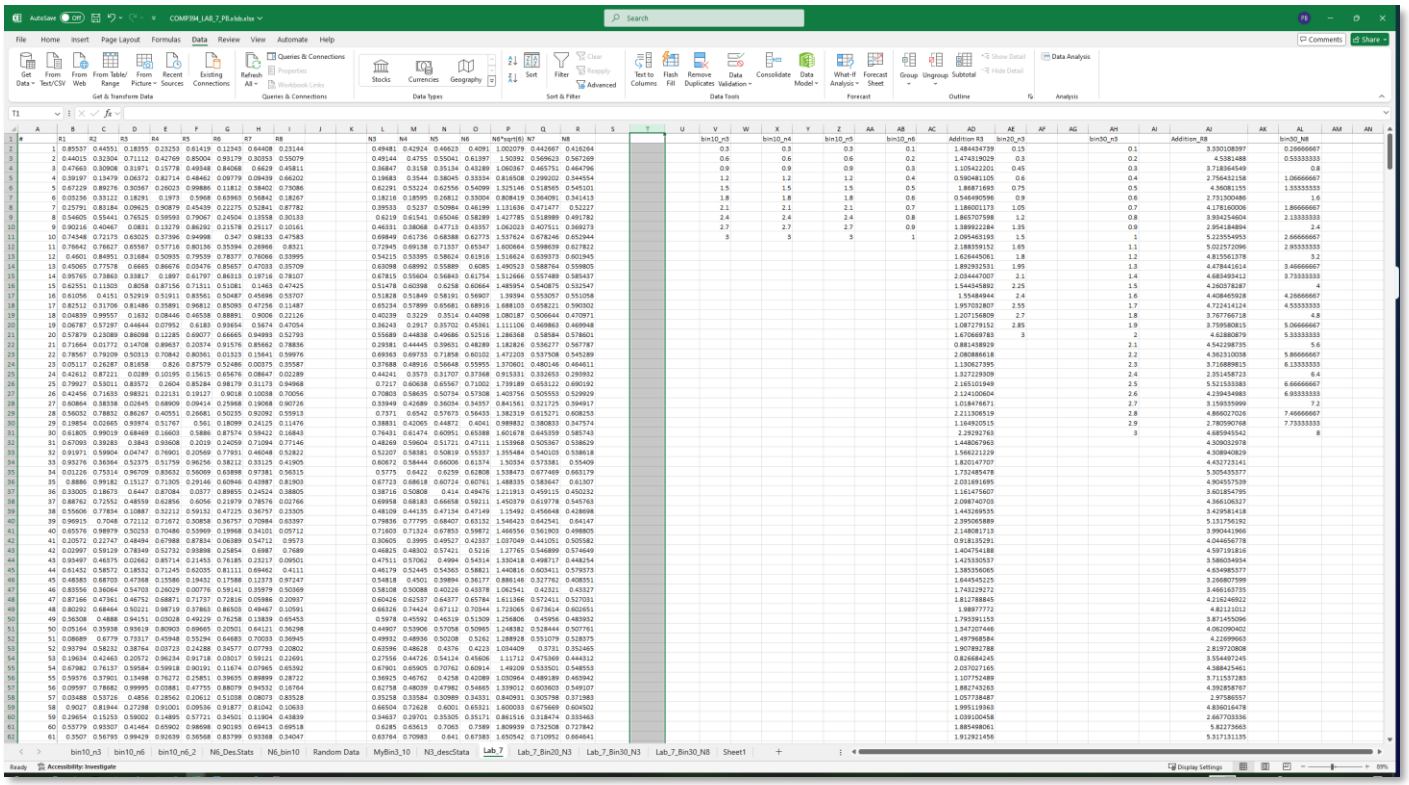


Fig 1.3: Added bin30_R8

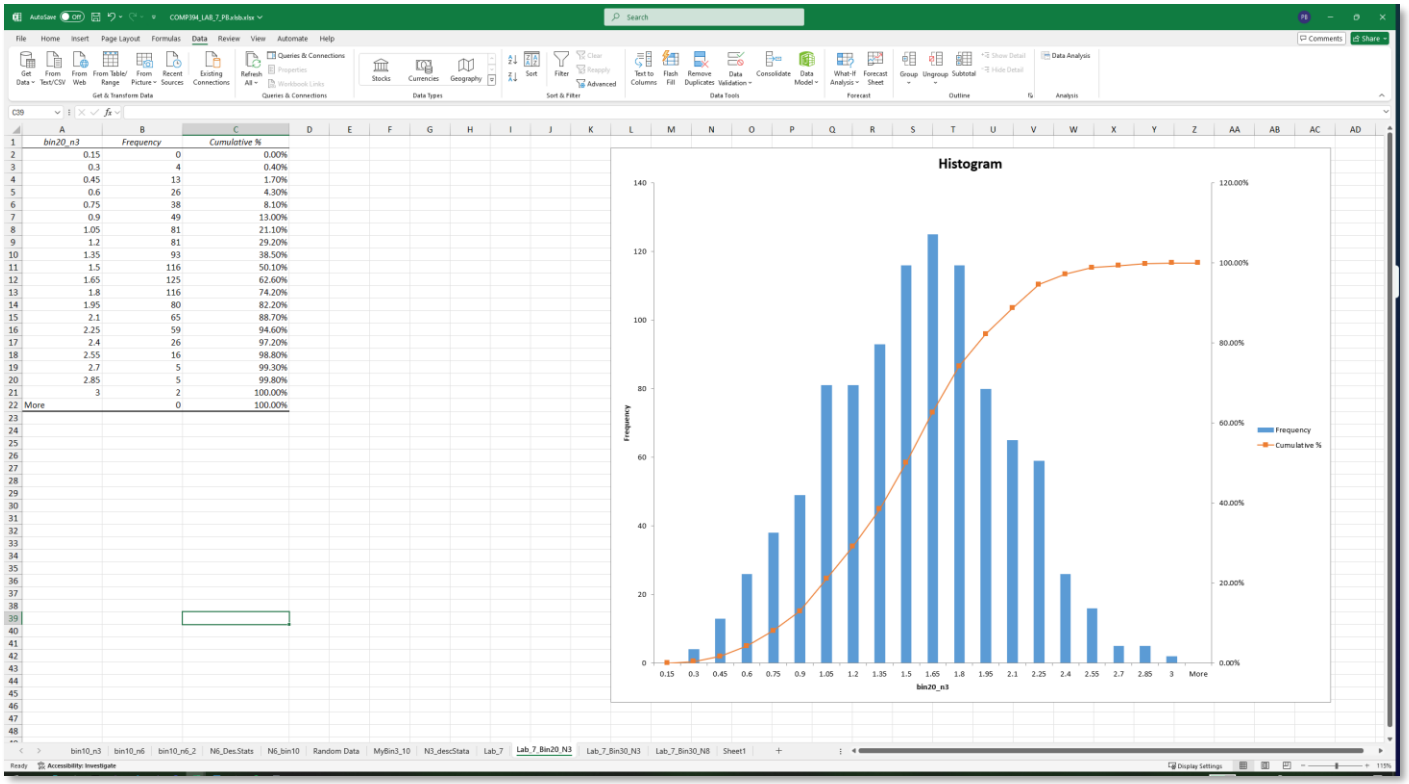


Fig 1.4: Plot of Bin20_n3

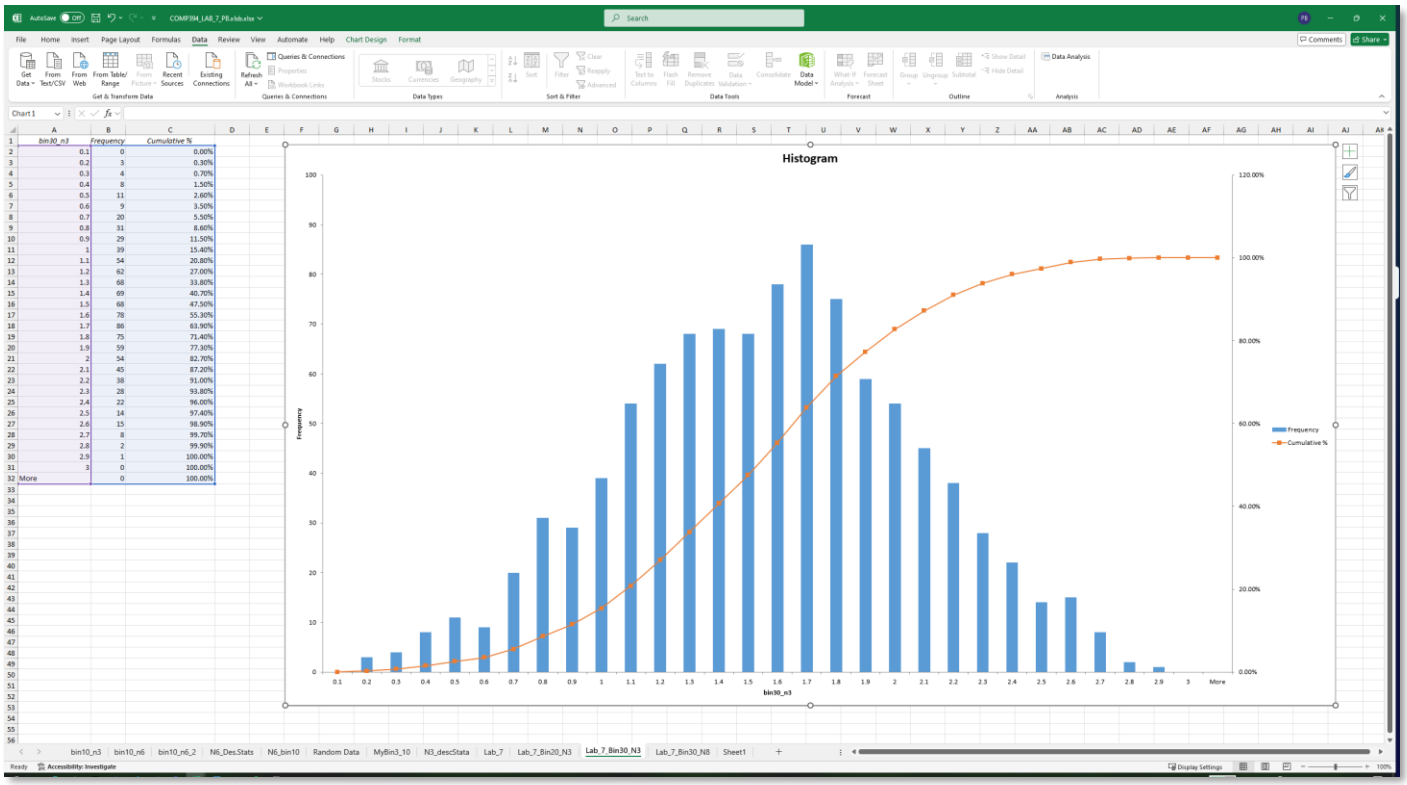


Fig 1.5: Plot of Bin30_n3

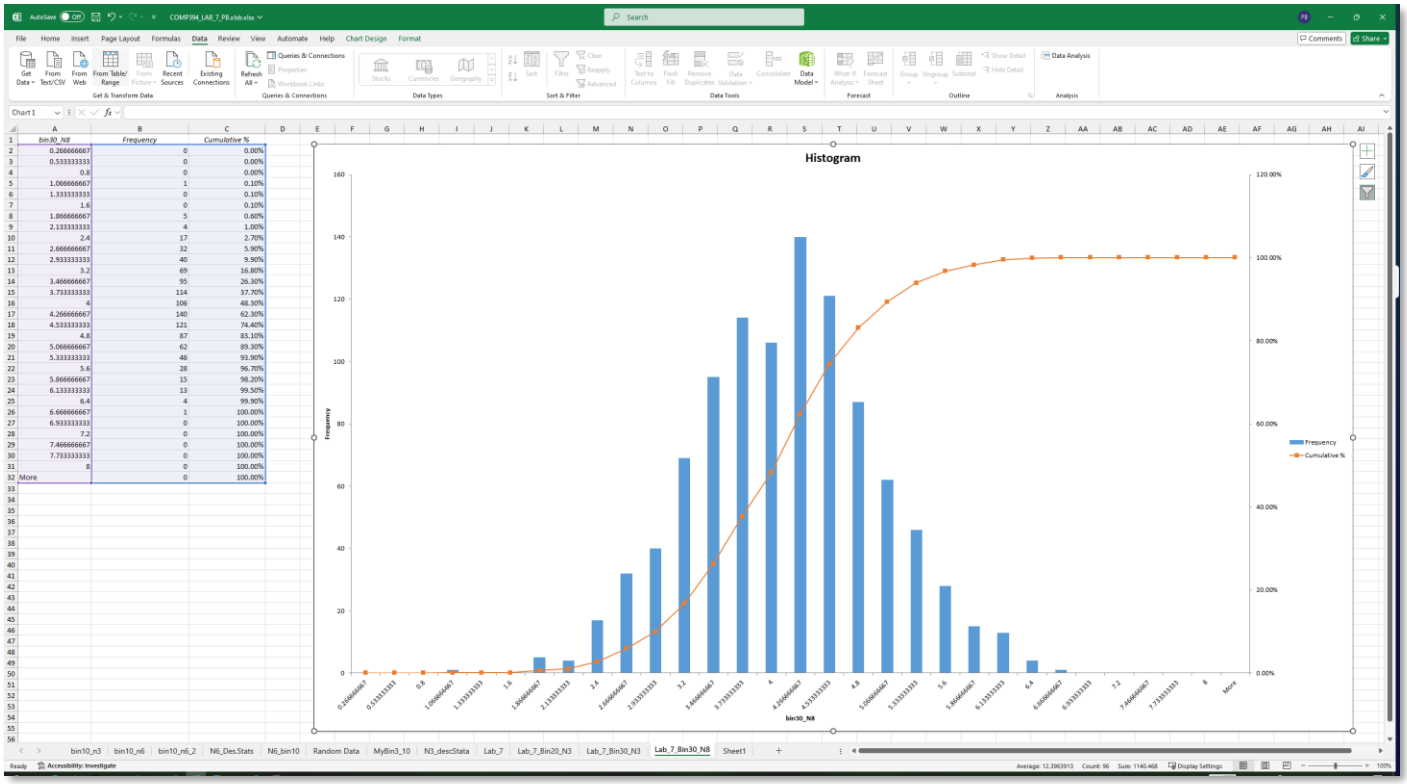


Fig 1.6: Plot of Bin30_N8

QUESTIONS:

1. Which number of bins works best, 10, 20 or 30?

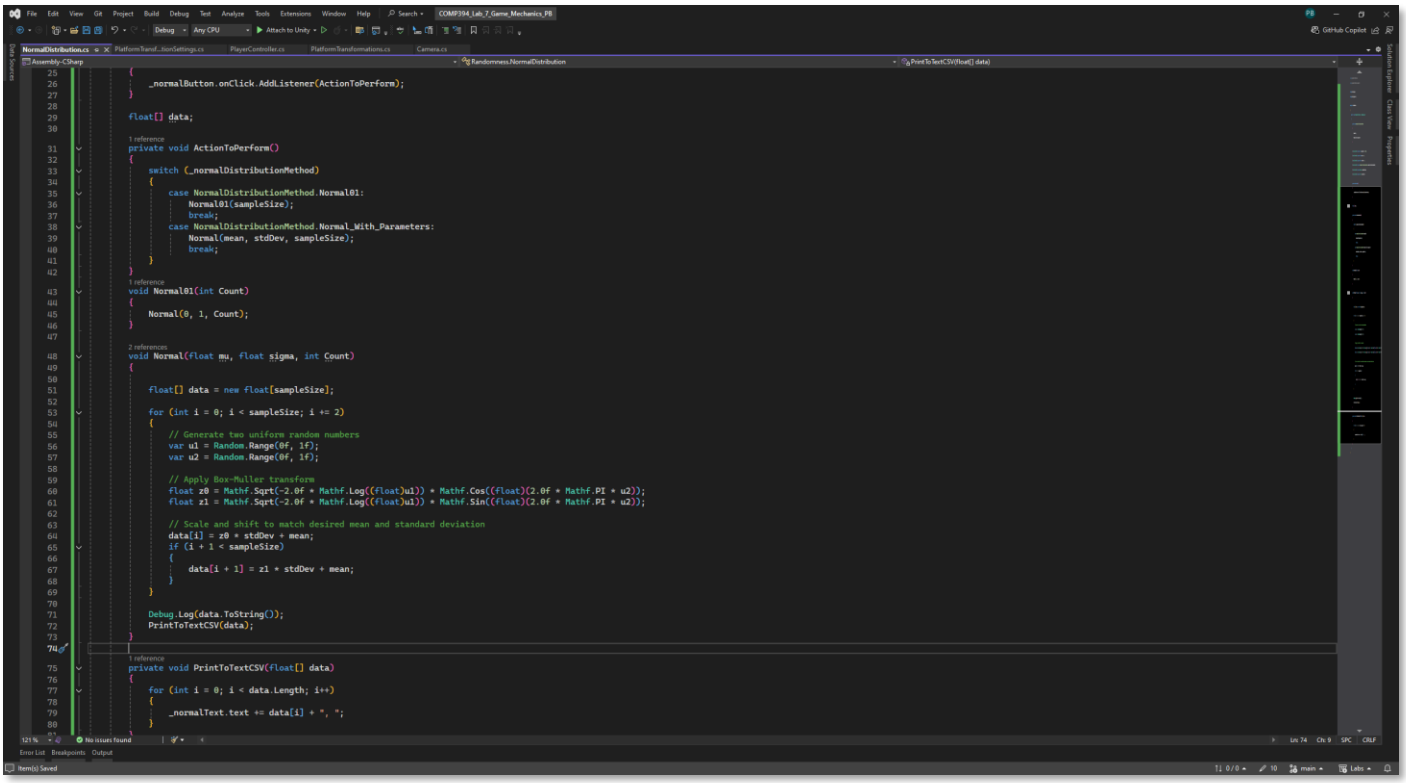
- *The bin with larger set works best because it shows more granular details and represents higher resolution of data. We can see the curve get smoother as we increase bins.*

2. Did you need to make corrections to start/end points of the bins, to get a better picture? If yes, what changes did you do?

- *No, the larger the bins the start/end points of the bins get evenly distributed which showcases the Bell curve. Also, as we go higher like in case of N8 we can se the start/end become smaller and smaller.*

3. Compare N3 with N8 in terms of proximity with normal distribution.

- *The most prominent difference can be seen at the ends of the curve. The ends of N8 Curve are shallower than the N3. Similarly, more chunk of data is present in the centre of the N8 graph than the N3 graph.*



```
    {
        _normalButton.onClick.AddListener(ActionToPerform);
    }

    float[] data;

    1 reference
    private void ActionToPerform()
    {
        switch (_normalDistributionMethod)
        {
            case NormalDistributionMethod.Normal01:
                Normal01(sampleSize);
                break;
            case NormalDistributionMethod.Normal_With_Parameters:
                Normal(mean, stdDev, sampleSize);
                break;
        }
    }
    1 reference
    void Normal01(int Count)
    {
        Normal(0, 1, Count);
    }

    2 references
    void Normal(float mu, float sigma, int Count)
    {

        float[] data = new float[sampleSize];

        for (int i = 0; i < sampleSize; i += 2)
        {
            // Generate two uniform random numbers
            var u1 = Random.Range(0f, 1f);
            var u2 = Random.Range(0f, 1f);

            // Apply Box-Muller transform
            float z0 = Mathf.Sqrt(-2.0f * Mathf.Log((float)u1)) * Mathf.Cos((float)(2.0f * Mathf.PI * u2));
            float z1 = Mathf.Sqrt(-2.0f * Mathf.Log((float)u1)) * Mathf.Sin((float)(2.0f * Mathf.PI * u2));

            // Scale and shift to match desired mean and standard deviation
            data[i] = z0 * stdDev + mean;
            if (i + 1 < sampleSize)
            {
                data[i + 1] = z1 * stdDev + mean;
            }
        }

        Debug.Log(data.ToString());
        PrintToTextCSV(data);
    }
    1 reference
    private void PrintToTextCSV(float[] data)
    {
        for (int i = 0; i < data.Length; i++)
        {
            _normalText.text += data[i] + ", ";
        }
```

Fig 1.7: C# Code for Normal Distribution

Fig 1.8: Output showing variables in normal distribution with count 500 for given parameters