Andrew Fryzel

Solo Final Project:

Procedurally Generated Terrain

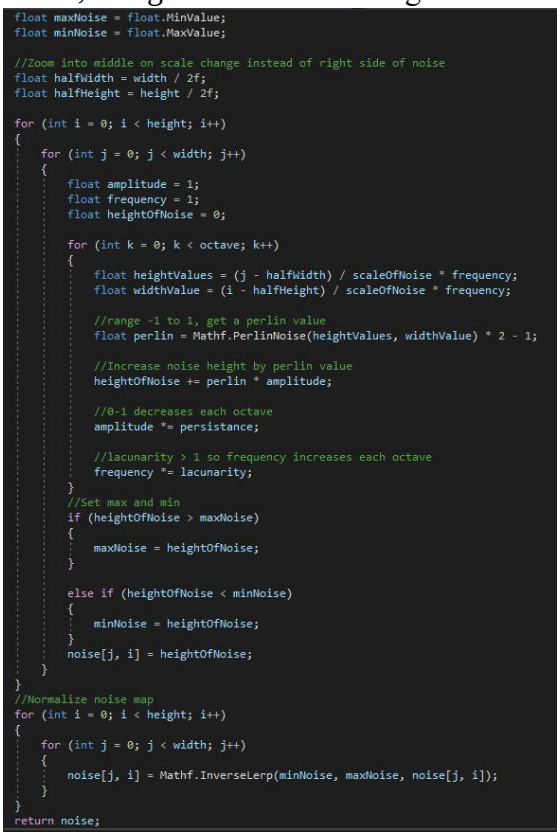
Video Link: https://youtu.be/LP7ggmH3EM8

For my final project, I looked into creating procedurally generated terrain using Perlin noise. Going into this project, I didn't have too much understanding on what most of these topics involved and so I had to do a lot of research and watch a lot of tutorial videos. All of my resources used can be found at the end of this document. I did all of the work myself and did not have any teammates.

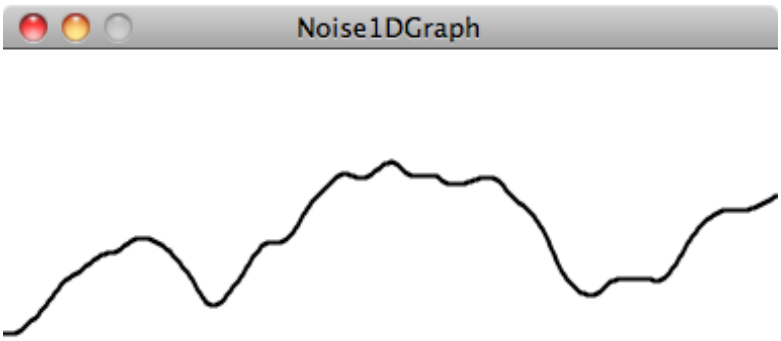
Procedurally Generated Terrain is a multi-step process and I had to learn each individual step. The steps involved include: Perlin Noise, Octaves, Frequency and Lacunarity, Meshing, and Character Movement (for the camera).

Perlin Noise

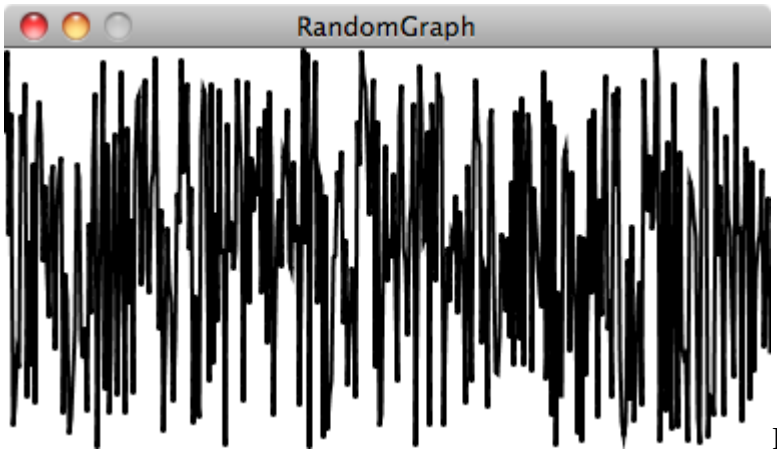
Perlin noise is at the center of most procedurally generated terrain techniques, fire, water, and other procedural functions. Perlin noise is a type of coherent noise, meaning that it changes gradually, which is perfect for procedural generation because of its smooth curve. Perlin noise can be used on 2D and 3D applications. Generating random numbers could accomplish the same purpose as Perlin noise, however, using a random number generation isn't as smooth or as clean as Perlin noise.



```
float maxNoise = float.MinValue;
float minNoise = float.MaxValue;

//Zoom into middle on scale change instead of right side of noise
float halfWidth = width / 2f;
float halfHeight = height / 2f;

for (int i = 0; i < height; i++)
{
    for (int j = 0; j < width; j++)
    {
        float amplitude = 1;
        float frequency = 1;
        float heightOfNoise = 0;

        for (int k = 0; k < octave; k++)
        {
            float heightValues = (j - halfWidth) / scaleOfNoise * frequency;
            float widthValue = (i - halfHeight) / scaleOfNoise * frequency;

            //range -1 to 1, get a perlin value
            float perlin = Mathf.PerlinNoise(heightValues, widthValue) * 2 - 1;

            //Increase noise height by perlin value
            heightOfNoise += perlin * amplitude;

            //0-1 decreases each octave
            amplitude *= persistance;

            //lacunarity > 1 so frequency increases each octave
            frequency *= lacunarity;
        }
        //Set max and min
        if (heightOfNoise > maxNoise)
        {
            maxNoise = heightOfNoise;
        }

        else if (heightOfNoise < minNoise)
        {
            minNoise = heightOfNoise;
        }
        noise[j, i] = heightOfNoise;
    }
}
//Normalize noise map
for (int i = 0; i < height; i++)
{
    for (int j = 0; j < width; j++)
    {
        noise[j, i] = Mathf.InverseLerp(minNoise, maxNoise, noise[j, i]);
    }
}
return noise;
```



Perlin Noise



Random Numbers

Source: https://cdn.kastatic.org/ka-perseus-images/0fd97fc7ab7ac5a7670935f1695d2a0c614e5252.png

https://cdn.kastatic.org/ka-perseus-images/81e9d201147cd09f1b78f9541993d8460355eb3e.png

https://www.khanacademy.org/computing/computer-programming/programming-natural-simulations/programming-noise/a/perlin-noise

Octaves, Frequency, Lacunarity

These three variables determine how the noise looks by changing different values of the noise. In a nutshell, octaves influence how many **levels** of detail the noise has; lacunarity determines the frequency of the octaves (the **detail** of each octave); persistence determines the amplitude, or the **shape**, of each octave; and the scale is the **zoom in / zoom out** factor of the Perlin noise. The octaves are combined together to form surfaces, or layers, of noise. It's easy to conceptualize these octave layers by thinking of the octaves as the levels on a topographic map (at least that's how I visualized it all).

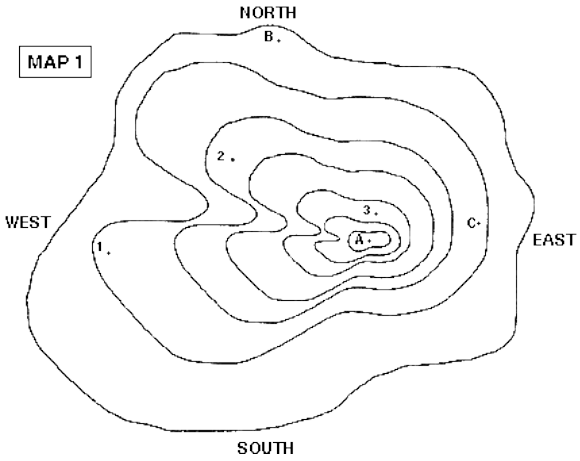


Image Source: https://i.pinimg.com/originals/56/4c/a7/564ca726d48c6b952f6141cf44f6f0a7.gif

```
//range -1 to 1, get a perlin value
float perlin = Mathf.PerlinNoise(heightValues, widthValue) * 2 - 1;

//Increase noise height by perlin value
heightOfNoise += perlin * amplitude;

//0-1 decreases each octave
amplitude *= persistance;

//lacunarity > 1 so frequency increases each octave
frequency *= lacunarity;
```

Coloring

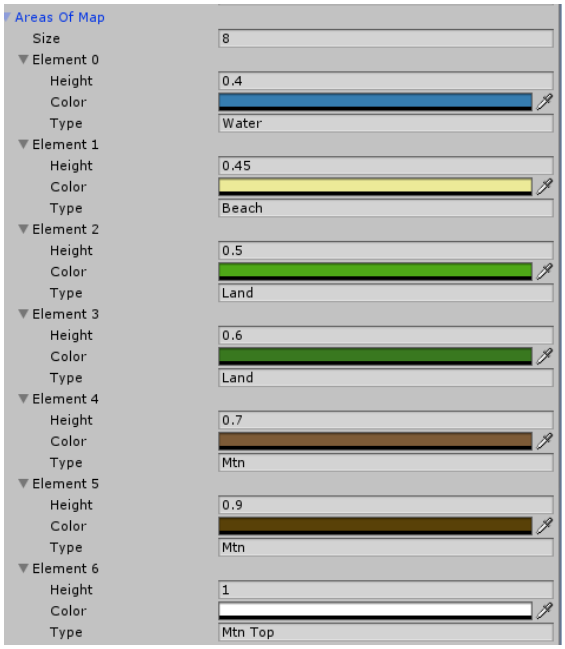
Coloring in Unity wasn't particularly difficult. I simply took the height values and assigned a color to any values within a certain range. I could have fine tuned the numbers, and added more colors, for a more realistic and visually appealing product. I searched through the heights, and then updated the colors based on the height of the noise.

```
//store the colors
Color[] colorOfAreas = new Color[width * height];
for (int i = 0; i < height; i++)
{
    for (int j = 0; j < width; j++)
    {
        float landHeight = noise[j, i];
        for (int k = 0; k < areasOfMap.Length; k++)
        {
            if (landHeight <= areasOfMap[k].height)
            {
                colorOfAreas[i * width + j] = areasOfMap[k].color;
                break;
            }
        }
    }
}
```

```
Texture2D texture = new Texture2D(width, height);
texture.SetPixels(colorOfAreas);
texture.Apply();

mapToDisplay.DrawColor(texture);
```

DrawColor Method

```
//Polish the edges and smooth it all out
texture.filterMode = FilterMode.Point;
texture.wrapMode = TextureWrapMode.Clamp;
//Render
renderTextures.sharedMaterial.mainTexture = texture;
renderTextures.transform.localScale = new Vector3(texture.width, 1, texture.height);
```



Mesh

Thankfully, this wasn't my first experience with meshing, and I understood the principles from previous projects and homework. The idea of meshing on this terrain map is somewhat complicated, but it involves calculating triangle vertices, performing vector calculations on those vertices and then normalizing the values. The hardest part of this section was determining how to connect it to Unity and map it work. I had to create several additional materials and scripts to get it to work, but in the end it did. Fundamentally, the triangles that are added to the mesh are at

1) (theVertexIndex, theVertexIndex + width + 1, theVertexIndex + width)

AND

2) (vertexIndex + width + 1, vertexIndex, vertexIndex + 1)

```
public void DrawMesh(MeshData mesh, Texture2D texture)
{
    //Polish the edges and smooth it all out
    texture.filterMode = FilterMode.Point;
    texture.wrapMode = TextureWrapMode.Clamp;
    //Render
    renderTextures.sharedMaterial.mainTexture = texture;
    //MeshFilter
    filter.sharedMesh = mesh.GenerateTheMesh();
    render.sharedMaterial.mainTexture = texture;
    renderTextures.transform.localScale = new Vector3(texture.width, 1, texture.height);
}
```

```
public void Triangle(int a, int b, int c)
{
    triangles[index] = a;
    triangles[index + 1] = b;
    triangles[index + 2] = c;
    index += 3;
}
```

Works Cited:

Map Generation Overview:

Procedural Terrain Generation

https://www.youtube.com/playlist?list=PLFt_AvWsXl0eBW2EiBtl_sxmDtSgZBxB3

Roguelike game in C++ - Map generation with Perlin noise:

https://solarianprogrammer.com/2012/07/25/roguelike-game-cpp-11-part-3/

How Procedurally Generated Terrain Works:

https://www.youtube.com/watch?v=JdYkcrW8FBg

Building an Infinite Procedurally-Generated World:

https://spin.atomicobject.com/2015/05/03/infinite-procedurally-generated-world/

Building an Infinite Procedurally-Generated World:

https://spin.atomicobject.com/2015/05/03/infinite-procedurally-generated-world/

Perlin Noise:

Playing with Perlin Noise: Generating Realistic Archipelagos

https://medium.com/@yvanscher/playing-with-perlin-noise-generating-realistic-archipelagos-b59f004d8401

How Perlin Noise Works:

http://www.huttar.net/lars-kathy/graphics/perlin-noise/perlin-noise.html

Programming Perlin-like Noise (C++):

https://www.youtube.com/watch?v=6-0UaeJBumA

Coding Challenge #11: 3D Terrain Generation with Perlin Noise in Processing:

https://www.youtube.com/watch?v=IKB1hWWedMk&vl=en

Making maps with noise functions:

https://www.redblobgames.com/maps/terrain-from-noise/

How can I tile Perlin noise to more accurately represent a world map?:

https://gamedev.stackexchange.com/questions/162223/how-can-i-tile-perlin-noise-to-more-accurately-represent-a-world-map

Game Mechanics:

How to make a Video Game in Unity - MOVEMENT (E03):

https://www.youtube.com/watch?v=Au8oX5pu5u4

Mesh:

Good resource on procedural mesh generation?

https://www.reddit.com/r/proceduralgeneration/comments/4zttau/good_resource_on_procedural_mesh_generation/

Procedural Mesh Generation using Unity Job System:

https://www.reddit.com/r/Unity3D/comments/8dajdl/procedural_mesh_generation_using_unity_job_system/

Procedural Mesh Geometry:

https://docs.unity3d.com/Manual/GeneratingMeshGeometryProcedurally.html

Procedural Grid

https://catlikecoding.com/unity/tutorials/procedural-grid/

how do i move a camera with mouse:

https://answers.unity.com/questions/1397655/how-do-i-move-a-camera-with-mouse.html

Rigidbody - how to stop it quickly:

https://answers.unity.com/questions/662811/rigidbody-how-to-stop-it-quickly.html

Input.GetKeyUp:

https://docs.unity3d.com/ScriptReference/Input.GetKeyUp.html?_ga=2.208990299.815134077.1576287530-645961870.1576198305