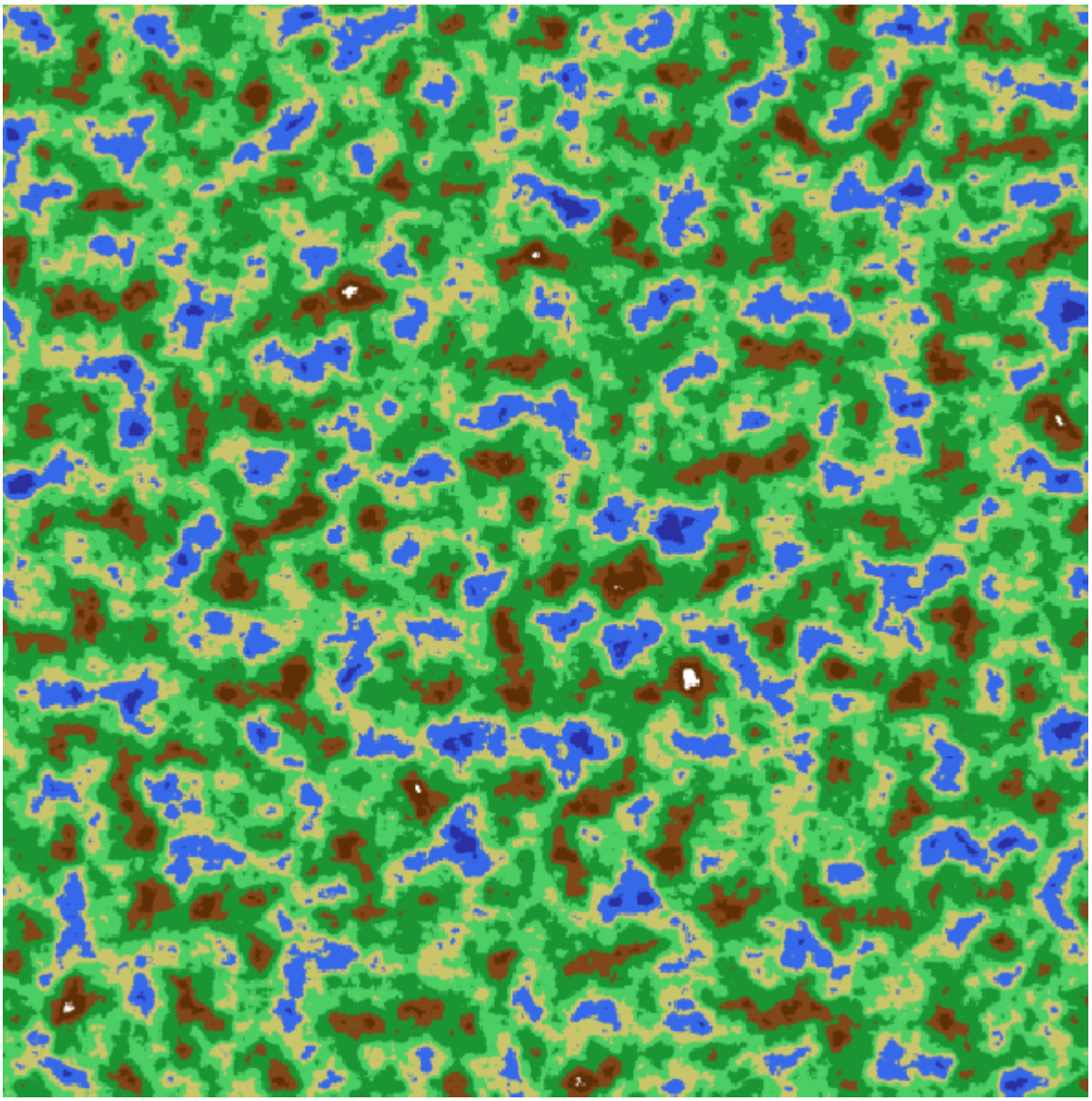
Genereren van een terreinmap aan de hand van Perlin Noise

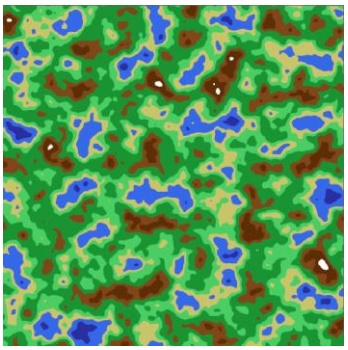
Jarno Vandeburie



Abstract

Perlin noise wordt vooral gebruikt om organische structuren en realistische landschappen te genereren. In dit document wordt overlopen hoe een kleurmap kan worden gegenereerd aan de hand van perlin noise. Deze map zal pseudorandom gegenereerd worden aan de hand van een seed.

We zullen deze kleurmap maken in de game engine Unity omdat dit een eenvoudige manier is en omdat kleurmappen vaak gebruikt worden om het terrein van games te maken.



Inhoudsopgave

Abstract 2

Figurenlijst 4

Introductie 5

Technische kennis 6

- Lacunarity 6
- Persistance 6

Maken van de noise map 7

- Perlin noice map 7
- Visualiseer de map 8
- Toevoegen van lacunarity en persistance 10
- Unieke noise maps 11

Noisemap omzetten naar een kleurmap 12

- Terreintypes aanmaken 12
- Implementatie terreintypes 12
- Codeduplicatie vermeiden 13
- Oplossen oneffenheden 14
- Voorbeeld kleurmap 14

Experimenteren in de editor 15

- Map width en map height 15
- Noise scale 15
- Octaven 15
- Lacunarity en persistente 15

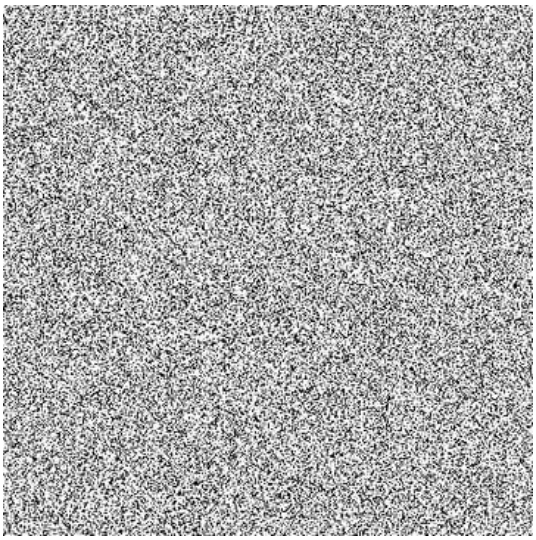
Bronvermelding 16

Figurenlijst

Figuur 1: Voorbeeld kleurmap 1
Figuur 2: Wat is een kleurmap 2
Figuur 3: Perlin noise map 5
Figuur 4: Regular noise map 5
Figuur 5: Octaven voor het toevoegen van lacunarity 6
Figuur 6: Octaven na het toevoegen van lacunarity 6
Figuur 7: Octaven voor het toevoegen van persistance 6
Figuur 8: Octaven na het toevoegen van persistance 6
Figuur 9: Inspector van het MapGenerator script 12
Figuur 10: Perlin noise map omgezet naar kleurmap 14
Figuur 11: Gegenereerde perlin noise map 14
Figuur 12: Inspector MapGenerator 15
Figuur 13: Lage scale gegenereerde map 15
Figuur 14: Hoge scale gegenereerde map 15

Introductie

Bij regular noise is elke pixel is een waarde tussen 0 en 1. Dit is random gegenereerd.
Bij perlin noise gebeurt dit iets vloeiender, wat zorgt voor grijstinten



Figuur 4: Regular noise map

Figuur 3: Perlin noise map

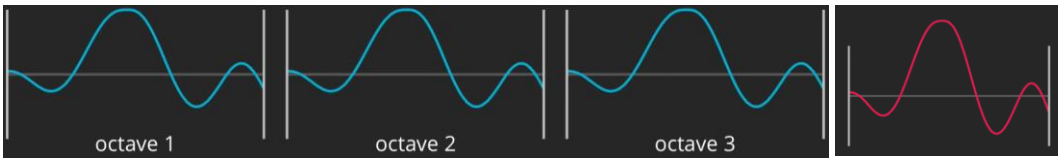
Als we hier een deel uit nemen, genaamd een octaaf, dan krijgen we iets wat lijkt op een gladde doorsnede van heuvelachtig landschap. Deze doorsnede is niet realistisch in vergelijking met de echte wereld dus hier moet er nog extra details aan worden toegevoegd. In het geval van het creëren van terrein kunnen we een octaaf gebruiken voor de outline van het terrein, een andere voor de grote rotsen en een derde voor de stenen. Een octaaf spreidt zich over een x-as en een y-as. Deze assen tonen respectievelijk de amplitude en de frequentie aan.

Technische kennis

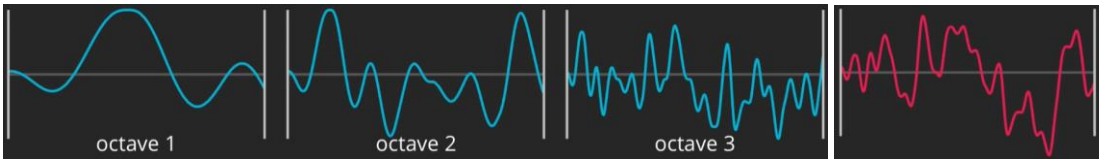
Lacunarity

Om de frequentie aan te passen voegen we een waarde toe genaamd de lacunarity. Het verhogen van de lacunarity verhoogt aantal details in de map. De frequentie van het eerste octaaf is de lacunarity tot de nulde macht, het tweede octaaf is de lacunarity tot de eerste macht, enz. Dit houdt dus in dat als we bijvoorbeeld 2 kiezen als waarde voor de lacunarity, dan is de frequentie van onze outline 1, de grote rotsen 2 en de stenen 4.

Hieronder is een voorbeeld weergegeven van hoe de lacunarity het eindresultaat van de doorsnede beïnvloedt.



lacunarity = 1

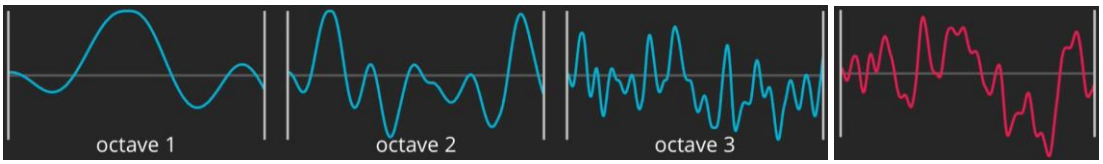


lacunarity = 2

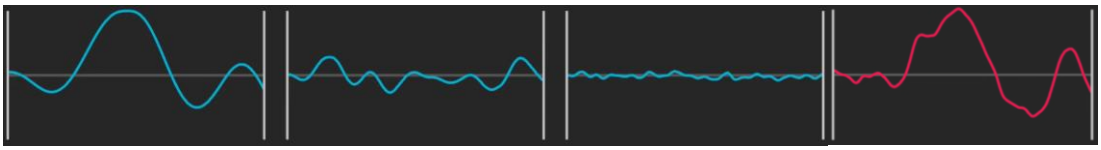
Persistance

De persistance is een waarde van 0 tot 1 en beïnvloedt hoe snel de amplitude verandert in elk octaaf. In tegenstelling tot de lacunarity verhoogt de persistance niet het aantal details maar hoe veel deze details het eindresultaat beïnvloeden. De persistance volgt dezelfde berekening als de lacunarity.

Dit zal ook de het eindresultaat beïnvloeden en zorgt samen met de resultaten van de lacunarity voor een realistische doorsnede.



persistance = 1



persistance = 0.5

Maken van de noise map

Perlin noice map

Een perlin noice map kan worden gemaakt aan de hand 2D-array van floats. Er wordt gekozen voor floats want de waarde van elke pixel ligt tussen 0 en 1. In een klasse, hier genaamd *Noise* wordt die array aangemaakt. Hierna gebruiken we de *Mathf.PerlinNoise* functie om een waarde te krijgen voor meegegeven coördinaten. We voegen ook nog een variabele *scale* toe aan de coördinaten. Als we dit niet doen, dan zouden we telkens dezelfde waarden krijgen voor dezelfde x-coördinaat en y-coördinaat. Deze variabele mag natuurlijk niet nul zijn want delen door nul zou een DivideByZeroException opleveren, dus we geven deze dan een standaardwaarde.

```
1.  using UnityEngine;
2.
3.  public static class Noise {
4.
5.      public static float[,] GenerateNoiseMap(int mapWidth, int mapHeight, float scale) {
6.          float[,] noiseMap = new float[mapWidth, mapHeight];
7.
8.          if (scale <= 0) {
9.              scale = 0.0001f;
10.         }
11.
12.         for (int y = 0; y < mapHeight; y++) {
13.             for (int x = 0; x < mapWidth; x++) {
14.                 float sampleX = x / scale;
15.                 float sampleY = y / scale;
16.
17.                 float perlinValue = Mathf.PerlinNoise(sampleX, sampleY);
18.                 noiseMap[x, y] = perlinValue;
19.             }
20.         }
21.         return noiseMap;
22.     }
23.
24. }
25.
```

Visualiseer de map

Om de perlin noice map te kunnen zien hebben we een leeg object nodig met een MapGenerator en een MapDisplay script. We beginnen met dit laatste script.

Het MapDisplay script zal de noise map omzetten naar een texture en zal dit texture toepassen op een plane. Om de map te kunnen weergeven moeten we elke pixel van de texture omzetten naar de bijhorende waarde uit de array. De snelste manier om dit te doen is om eerst een array van alle kleuren.

Hierna kunnen we de nieuwe texture toepassen. In onderstaande code doen we dit door de mainTexture van het sharedMaterial aan te passen. Dit zorgt ervoor dat we niet altijd in play-mode moeten gaan om de veranderingen te zien.

```
1.  using UnityEngine;
2.
3.  public class MapDisplay : MonoBehaviour {
4.      public Renderer textureRender;
5.
6.      public void DrawNoiseMap(float[,] noiseMap) {
7.          int width = noiseMap.GetLength(0);
8.          int height = noiseMap.GetLength(1);
9.
10.         Texture2D texture = new Texture2D(width, height);
11.
12.         Color[] colourMap = new Color[width * height];
13.         for (int y = 0; y < height; y++) {
14.             for (int x = 0; x < width; x++) {
15.                 colourMap[y * width + x] = Color.Lerp(Color.black, Color.white, noiseMap[x, y]);
16.             }
17.         }
18.
19.         texture.SetPixels(colourMap);
20.         texture.Apply();
21.
22.         textureRender.sharedMaterial.mainTexture = texture;
23.         textureRender.transform.localScale = new Vector3(width, 1, height);
24.     }
25. }
26.
```

Om de map te laten genereren gebruiken we jet MapGenerator script. Hierin hebben we een variabele voor de breedte, de hoogte en de noise scale van de map die we kunnen aanpassen in de editor. We beginnen met het aanmaken van de noise map via de eerder gecreëerde klasse *Noise* en de variabelen als parameters. Met de aangemaakte map kunnen we onze net gemaakte functie oproepen

```
1.  using UnityEngine;
2.
3.  public class MapGenerator : MonoBehaviour {
4.      public int mapWidth;
5.      public int mapHeight;
6.      public float noiseScale;
7.
8.      public void GenerateMap() {
9.          float[,] noiseMap = Noise.GenerateNoiseMap(mapWidth, mapHeight, noiseScale);
10.
11.         MapDisplay display = FindObjectOfType<MapDisplay>();
12.         display.DrawNoiseMap(noiseMap);
13.     }
14. }
15.
```

De laatste paar stappen die we moeten zetten is het opzetten van de objecten. Om te beginnen plaatsen we het plane-object in de Texture Render van het MapDisplay script van het MapGenerator object. Hierna maken we een nieuw Unlit Texture materiaal aan en voegen we dit toe aan onze plane.

Als laatste zullen we een knop toevoegen aan de MapGenerator in de editor. Dit doen we in een nieuw script genaamd. Dit moet in een apart bestand gebeuren want het script is geen interface van MonoBehaviour maar van Editor.

```
1.  using UnityEngine;
2.  using UnityEditor;
3.
4.  [CustomEditor (typeof (MapGenerator))]
5.  public class MapGeneratorEditor : Editor {
6.
7.      public override void OnInspectorGUI() {
8.          MapGenerator = (MapGenerator) target;
9.
10.         DrawDefaultInspector();
11.
12.         if (GUILayout.Button("Generate")) {
13.             mapGenerator.GenerateMap();
14.         }
15.     }
16. }
17.
```

Nu kunnen we in de editor de waarden aanpassen en een noise map genereren.

Toevoegen van lacunarity en persistance

Zoals vermeld in de introductie kunnen we lacunarity en persistance toevoegen aan de octaven die de noise bepalen. Dit doen we dan ook in het *Noise* script. Om een beter resultaat te bekomen zou de *perlinValue* af en toe een negatieve waarde moeten hebben. Bij gevolg wordt de *noiseHeight* ook negatief, maar omdat *Mathf.PerlinNoise* altijd een waarde tussen 0.0 en 1.0 teruggeeft, vermenigvuldigen we dit met 2 en trekken dan 1 af. Hierdoor zullen alle waarden lager dan 0.5 negatief worden.

Uiteindelijk moeten alle waarden in de noise map wel tussen 0 en 1 liggen. Om dit te doen houden we de minimum en de maximumwaardes bij. Daarna overlopen we nog eens alle waarden en zorgen we dat alle waarden tussen 0 en 1 liggen met behulp van *InverseLerp*.

InverseLerp is een functie met 3 parameters: een minimum, een maximum en waarde. Als de waarde gelijk is aan het minimum, dan geeft de functie 0 terug. Als de waarde gelijk is aan het maximum, dan geeft de functie 1 terug.

```
5.  public static float[,] GenerateNoiseMap(int mapWidth, int mapHeight, float scale, int octaves, float
    persistance, float lacunarity) {
6.          float[,] noiseMap = new float[mapWidth, mapHeight];
7.
8.          if (scale <= 0) {
9.              scale = 0.0001f;
10.         }
11.
12.         float maxNoiseHeight = float.MinValue;
13.         float minNoiseHeight = float.MaxValue;
14.
15.         for (int y = 0; y < mapHeight; y++) {
16.             for (int x = 0; x < mapWidth; x++) {
17.
18.                 float amp = 1;
19.                 float freq = 1;
20.                 float noiseHeight = 0;
21.
22.                 for (int i = 0; i < octaves; i++) {
23.                     float sampleX = x / scale * freq;
24.                     float sampleY = y / scale * freq;
25.
26.                     float perlinValue = Mathf.PerlinNoise(sampleX, sampleY) * 2 - 1;
27.                     noiseHeight += perlinValue * amp;
28.                     noiseMap[x, y] = perlinValue;
29.
30.                     amp *= persistance;
31.                     freq *= lacunarity;
32.                 }
33.
34.                 if (noiseHeight > maxNoiseHeight) {
35.                     maxNoiseHeight = noiseHeight;
36.                 } else if (noiseHeight < minNoiseHeight) {
37.                     minNoiseHeight = noiseHeight;
38.                 }
39.                 noiseMap[x, y] = noiseHeight;
40.             }
41.         }
42.
43.         for (int y = 0; y < mapHeight; y++) {
44.             for (int x = 0; x < mapWidth; x++) {
45.                 noiseMap[x, y] = Mathf.InverseLerp(minNoiseHeight, maxNoiseHeight, noiseMap[x, y]);
46.             }
47.         }
48.
49.         return noiseMap;
50.     }
51.
```

Natuurlijk moeten de nieuwe parameters ook ingevuld worden. Daarom maken we 3 nieuwe variabelen in de MapGenerator en voegen we ze toe aan de functie

```
8.  public int octaves;
9.      public float persistance;
10.     public float lacunarity;
```

```
11.
12.     public void GenerateMap() {
13.         float[,] noiseMap = Noise.GenerateNoiseMap(mapWidth, mapHeight, noiseScale, octaves,
    persistance, lacunarity);
14.
15.         MapDisplay display = FindObjectOfType<MapDisplay>();
16.         display.DrawNoiseMap(noiseMap);
17.     }
18.
```

Unieke noise maps

Het is natuurlijk de bedoeling om allerlei unieke noise maps te maken. Dit kunnen we doen aan de hand van de klasse *Random*. Om de map niet helemaal willekeurig te maken zodat we bepaalde zaken kunnen berekenen als we dat zouden willen, voegen we een *seed* toe als parameter.

```
5.  public static float[,] GenerateNoiseMap(int mapWidth, int mapHeight, float scale, int octaves, float
    persistance, float lacunarity, int seed) {
6.          float[,] noiseMap = new float[mapWidth, mapHeight];
7.
8.          System.Random random = new System.Random(seed);
9.          Vector2[] octaveOffsets = new Vector2[octaves];
10.         for (int i = 0; i < octaves; i++) {
11.             float offsetX = random.Next(-100000, 100000);
12.             float offsetY = random.Next(-100000, 100000);
13.             octaveOffsets[i] = new Vector2(offsetX, offsetY);
14.         }
15.
```

```
31. float sampleX = x / scale * freq + octaveOffsets[i].x;
32. float sampleY = y / scale * freq + octaveOffsets[i].y;
```

Vergeet ook zeker niet om de seed-variabele toe te voegen aan de MapGenerator.

Noisemap omzetten naar een kleurmap

Terreintypes aanmaken

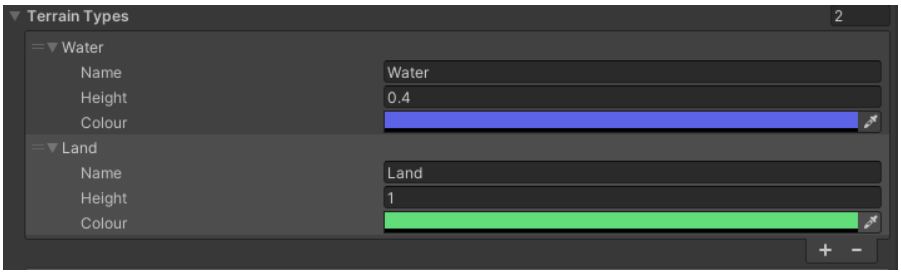
Nu we een willekeurig gegenereerde noise map hebben kunnen we deze gebruiken voor terrein te genereren. Dit kan door middel van kleur toe te kennen aan bepaalde *noiseHeight*-waarden. In het MapGenerator script voegen we een kleine struct toe onder de klasse. Een struct bepaald de variabelen dat een object heeft. We maken dit Serializable zodat we deze kunnen aanmaken en aanpassen in de editor.

```
23. [System.Serializable]
24. public struct TerrainType {
25.     public string name;
26.     public float height;
27.     public Color colour;
28. }
29.
```

Momenteel is het wel mogelijk om terreintypes aan te maken maar er is nog nergens een plaats om dit te doen. Als oplossing hiervoor maken we een variabele aan in de klasse van MapGenerator.

```
14. public TerrainType[] terrainTypes;
```

We kunnen nu onze eigen types aanmaken in de editor aan de hand van de GUI. Op onderstaande screenshot zijn is er een water en een land type gemaakt. Alle noiseheight-waarden van 0 tot en met 0.4 zal worden blauw gekleurd en alles boven 0.4 tot en met 1 worden groen gekleurd.



Figuur 9: Inspector van het MapGenerator script

Implementatie terreintypes

Om dit te implementeren zullen we voor elke noiseheight over alle types loopen en van zodra dat de noiseheight-waarde kleiner is dan de waarde van het type, dan hebben we de het bijhorend type gevonden. Hierna doen we hetzelfde als bij in het MapDisplay script, namelijk onze 2D noisemap omzetten naar een 1D kleurmap. Omdat het gemakkelijk zou zijn om zowel de noisemap als de kleurmap te kunnen zien, zullen we ook een enum toevoegen zodat we kunnen kiezen wat we zien.

```
5.  public enum DrawMode { NoiseMap, ColourMap }
6.  public DrawMode drawMode;
7.
```

```
1.  public void GenerateMap() {
2.          float[,] noiseMap = Noise.GenerateNoiseMap(mapWidth, mapHeight, noiseScale, octaves,
                persistance, lacunarity, seed);
3.
4.          Color[] colourMap = new Color[mapWidth * mapHeight];
5.          for (int y = 0; y < mapHeight; y++) {
6.              for (int x = 0; x < mapWidth; x++) {
7.                  for (int i = 0; i < terrainTypes.Length; i++) {
8.                      if (noiseMap[x, y] <= terrainTypes[i].height) {
9.                          colourMap[y * mapWidth + x] = terrainTypes[i].colour;
10.                         break;
11.                     }
12.                 }
13.             }
```

```
        }

        MapDisplay display = FindObjectOfType<MapDisplay>();

        if (drawMode == DrawMode.NoiseMap) {
            display.DrawNoiseMap(noiseMap);
        } else if (drawMode == DrawMode.ColourMap) {
            display.DrawNoiseMap(colourMap);
        }
    }

```

Codeduplicatie vermeiden

Zoals vermeld is dit een duplicatie van code en dat is slecht voor de leesbaarheid en de verstaanbaarheid van de code. Dit kunnen we oplossen door een script genaamd TextureGenerator aan te maken. Hierin kunnen we dan bovenstaande code omvormen tot een meer bruikbare functie. De code kan nog verder gesplitst worden in het maken van een map, dit zullen we doen met *TextureFromHeightMap*, en het inladen van de map, dit zullen we dan weer met *TextureFromColourMap* doen.

```
using UnityEngine;

public static class TextureGenerator {

    public static Texture2D TextureFromColourMap(Color[] colourMap, int width, int height) {
        Texture2D texture = new Texture2D(width, height);
        texture.SetPixels(colourMap);
        texture.Apply();
        return texture;
    }

    public static Texture2D TextureFromHeightMap(float[,] heightMap) {
        int width = heightMap.GetLength(0);
        int height = heightMap.GetLength(1);

        Color[] colourMap = new Color[width * height];
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                colourMap[y * width + x] = Color.Lerp(Color.black, Color.white, heightMap[x, y]);
            }
        }

        return TextureFromColourMap(colourMap, width, height);
    }
}

```

Dit zorgt ook voor het properder maken van de code in MapDisplay en het aanpassen van een blok code in MapGenerator.

```
using UnityEngine;

public class MapDisplay : MonoBehaviour
{
    public Renderer textureRender;

    public void DrawTexture(Texture2D texture) {
        textureRender.sharedMaterial.mainTexture = texture;
        textureRender.transform.localScale = new Vector3(texture.width, 1, texture.height);
    }
}

```

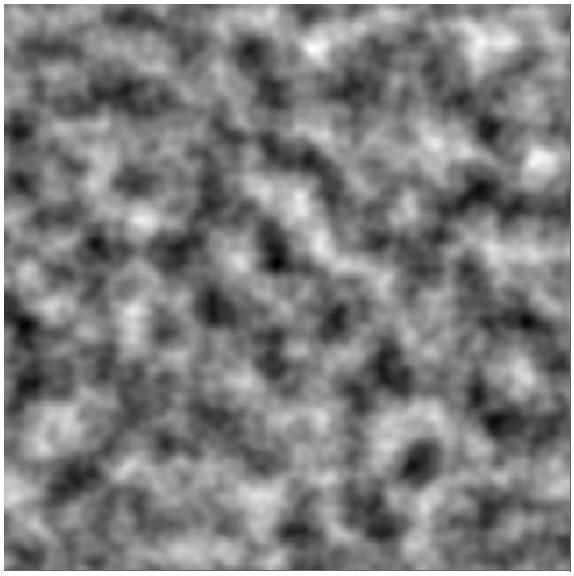
```
if (drawMode == DrawMode.NoiseMap) {
            display.DrawTexture(TextureGenerator.TextureFromHeightMap(noiseMap));
        } else if (drawMode == DrawMode.ColourMap) {
            display.DrawTexture(TextureGenerator.TextureFromColourMap(colourMap, mapWidth, mapHeight));
        }

```

Oplossen oneffenheden

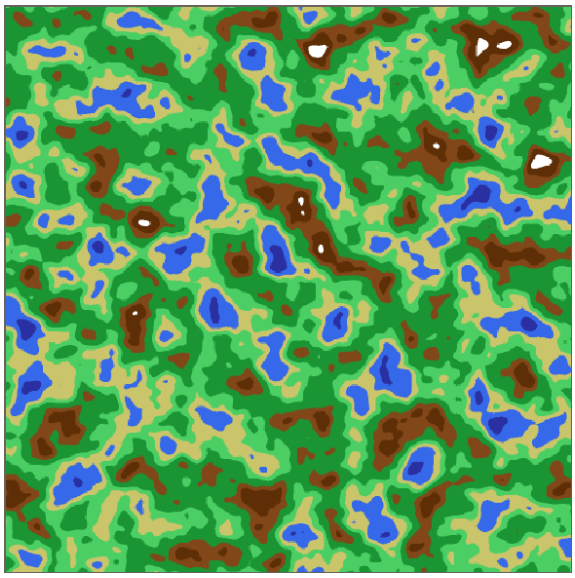
Als we inzoomen vallen direct wat oneffenheden op: ten eerste is de overhang van terreintypes wazig en ten tweede zijn de zijkanten incorrect. Dit is omdat de wrap-optie van het texture standaard 'herhalend' is en zo zie je een klein beetje van de overkant van de map. Dit is beiden makkelijk op te lossen door onderstaande code toe te voegen aan de TextureFromColourMap-functie in TextureGenerator.

```
9.  texture.filterMode = FilterMode.Point;
10. texture.wrapMode = TextureWrapMode.Clamp;
```

Voorbeeld kleurmap

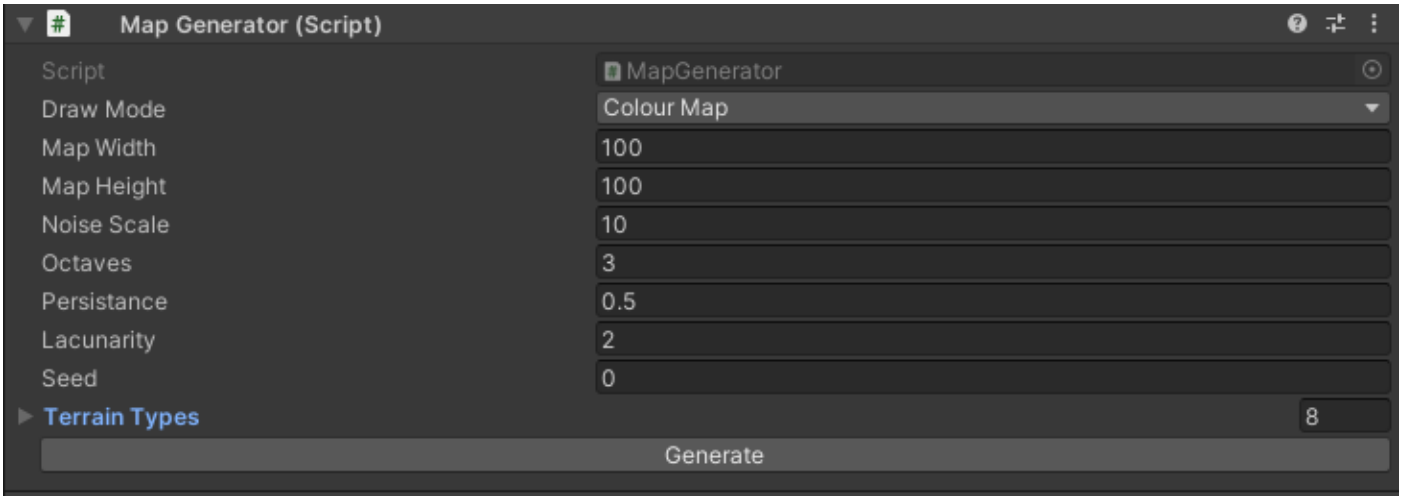


Figuur 11: Gegenereerde perlin noise map



Figuur 10: Perlin noise map omgezet naar kleurmap

Experimenteren in de editor



Figuur 12: Inspector MapGenerator

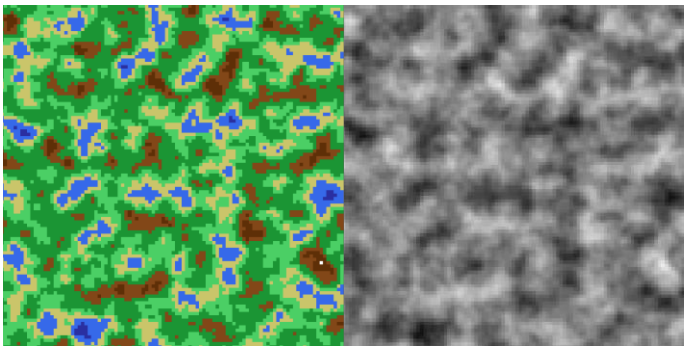
Het MapGenerator script zorgt dat we veel variabelen hebben, maar om het beoogde resultaat te behalen kan het wat zoeken zijn om de juiste waarden te vinden.

Map width en map height

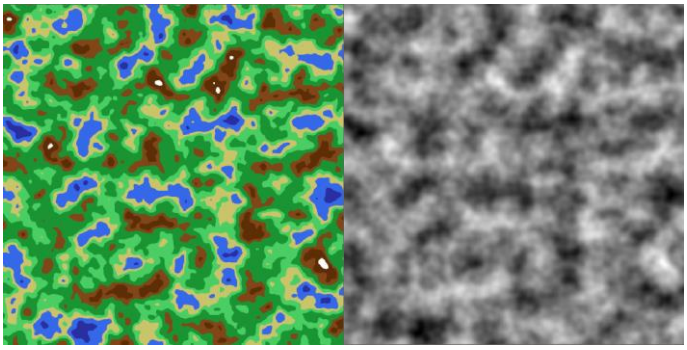
Dit is hoe groot de map is als deze gegenereerd wordt. Het is aangeraden om geen astronomische waarden hieraan toe te kennen want dit kan een eind duren en mogelijks ook het programma of de computer laten crashen.

Noise scale

De noise scale kan worden vergeleken met zoomen. Als de map width en de map height gelijk blijven, dan zal een hogere noise scale zorgen voor meer details en een lagere noise scale zal een grotere map genereren, maar met veel minder details. Onderstaande afbeeldingen zijn twee screenshots van eenzelfde regio op een map. Op de eerste afbeelding heeft de noise scale een waarde van 10 en op de tweede afbeelding een waarde van 100. Met een noise scale waarde van 10 was de map wel 100 keer groter.



Figuur 14: Lage scale gegenereerde map



Figuur 13: Hoge scale gegenereerde map

Octaven

Naast de noise scale is er een factor die nog meer het detaillevel bepaald, namelijk de octaven. Hoe hoger het aantal octaven, hoe gedetailleerder de map zal zijn.

Lacunarity en persistente

Zoals vermeld in de introductie heeft zowel de lacunarity als de persistance een grote invloed op het eindresultaat. De lacunarity bepaald hoe veel detail er is toegevoegd aan elk octaaf en zal dus de frequency aanpassen. De persistance zal dan weer bepalen hoe veel effect elk octaaf heeft op het eindresultaat en zal dus de amplitude aanpassen.

Bronvermelding

Scher, Y. (2020, 24 juni). *Playing with Perlin Noise: Generating Realistic Archipelagos*. Medium. Geraadpleegd op 10 februari 2022, van https://medium.com/@yvanscher/playing-with-perlin-noise-generating-realistic-archipelagos-b59f004d8401

Sebastian Lague. (2016, 31 januari). *Procedural Terrain Generation*. YouTube. Geraadpleegd op 8 februari 2022, van https://www.youtube.com/playlist?list=PLFt_AvWsXl0eBW2EiBtl_sxmDtSgZBxB3