

Laboratorium ZTPGK

# **Temat: Prosta implementacja renderera terenu**

Prowadzący: dr inż. Michał Staniszewski

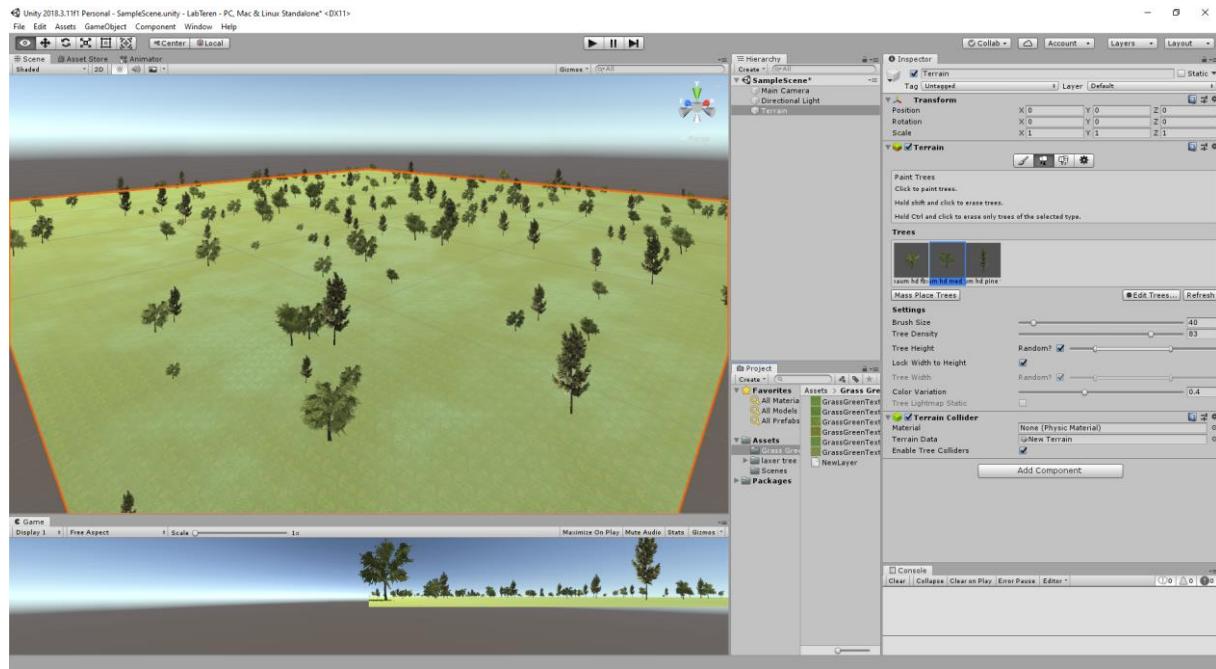
Data przeprowadzenia laboratorium: 13.03.2020 r.

**IGT Inf sem1**

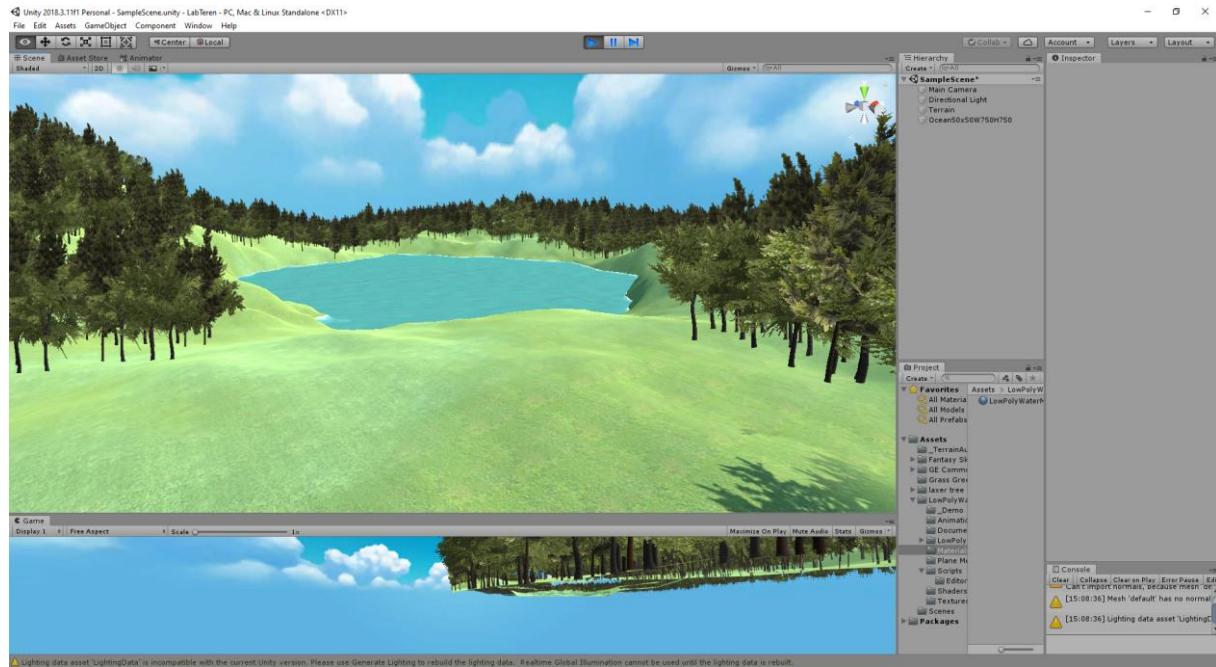
*Karol Jaźdrzyk*

## 1. Tworzenie prostego terenu

W pierwszej kolejności zapoznano się z możliwościami tworzenia prostego terenu z poziomu edytora Unity. Dodana została tekstura, oświetlenie wraz z flarami oraz proste drzewa.



Następnie, został ukształtowany ręcznie teren przy pomocy narzędzia Paint Terrain. Skybox został zamieniony na inny oraz dodano prostą wodę imitującą jezioro. Liczbę drzew zwiększo do 1000 i ręcznie usunięto nadmiarowe tak, aby uzyskać zadowalający rezultat.



## 2. Symulacja terenu

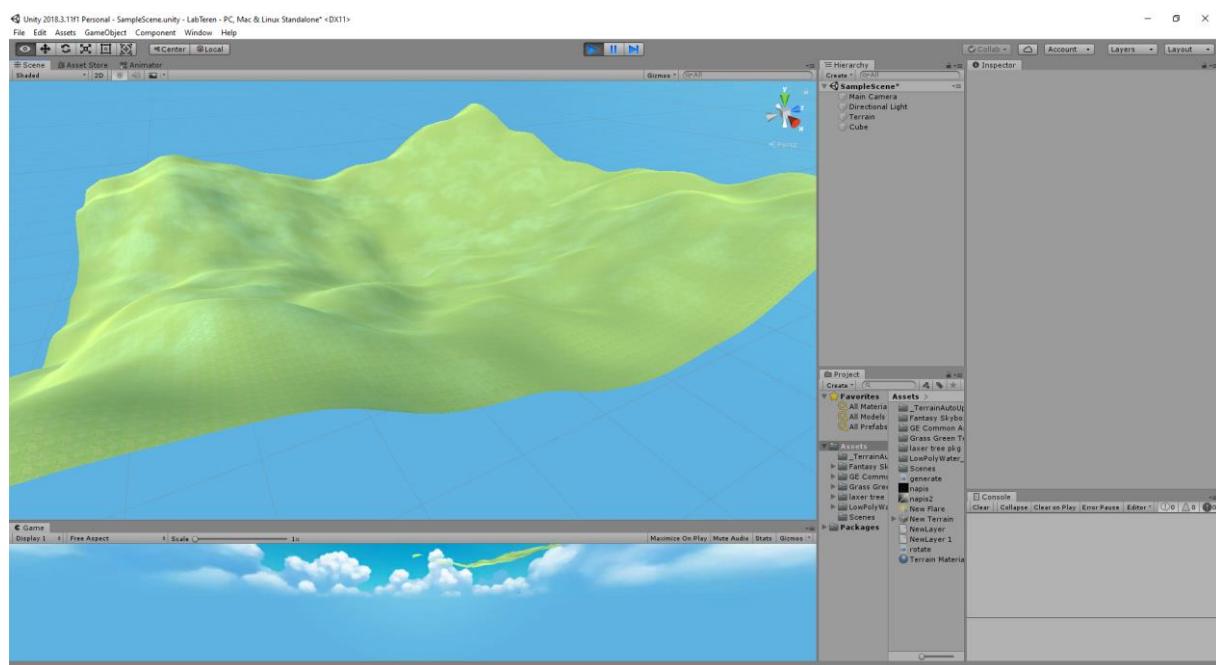
W następnej kolejności utworzono na scenie sześciian, a w nim skrypt o nazwie *rotate* powodujący jego obrót w celu sprawdzenia działania skryptów. Sześciian obracał się zgodnie z założeniami. Później została utworzony skrypt *generate* w obiekcie Terrain znajdującym się na scenie. Skopiowano do niego zawartość zadania 4 z instrukcji. Metodę do generowania bazowego terenu zmodyfikowano poprzez dodanie nakładających się, różnych amplitud i zakresów skali dla szumu Perlin'a:

```

float[] scale = new float[numberOfPasses];
float[] scale2 = new float[numberOfPasses];
float[] scale3 = new float[numberOfPasses];
for (int k = 0; k < numberOfPasses; k++)
{
    scale[k] = UnityEngine.Random.Range(0.0f, 3.0f);
    scale2[k] = UnityEngine.Random.Range(0.0f, 20.0f);
    scale3[k] = UnityEngine.Random.Range(5.0f, 10.0f);
}
for (int i = 0; i < _xRes; i++)
{
    for (int j = 0; j < _yRes; j++)
    {
        float xCoeff = (float)i / _xRes;
        float yCoeff = (float)j / _yRes;
        _terrHeights[i, j] = 0;
        for (int k = 0; k < numberOfPasses; k++)
        {
            _terrHeights[i, j] += Mathf.PerlinNoise(xCoeff * scale[k],
yCoeff * scale[k])*0.5f;
            _terrHeights[i, j] += Mathf.PerlinNoise(xCoeff * scale2[k],
yCoeff * scale2[k])*0.05f;
            _terrHeights[i, j] += Mathf.PerlinNoise(xCoeff * scale3[k],
yCoeff * scale3[k]) * 0.05f;
        }
        _terrHeights[i, j] /= (float)numberOfPasses;
    }
}

```

Wartości dobrano tak, aby uzyskać rezultat imitujący zadowalający górzysty teren:



W metodzie *AnimTerrain()* odpowiedzialnej za symulacje zostały dodane zmienne X oraz Y odpowiedzialne za umiejscowienie animacji na powierzchni terenu:

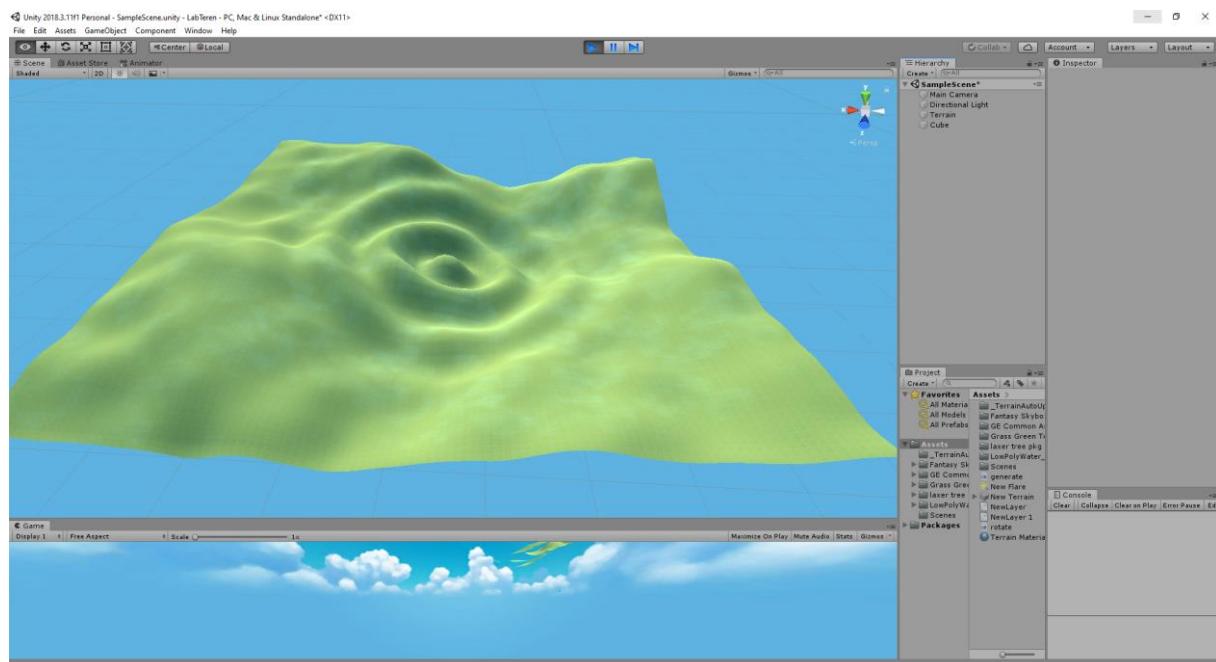
|                     |    |
|---------------------|----|
| X                   | 50 |
| Y                   | 50 |
| Number Of Passes    | 3  |
| Radius Of Animation | 20 |

Odchylenie od pierwotnej wysokości zostało przeskalowane aby uzyskać subtelniejszy (przyjemniejszy dla oka) ruch na powierzchni terenu. Kod oraz wyniki poniżej:

```
private void AnimTerrain()
{
    _terrHeights = _myTerrData.GetHeights(x, y, radiusOfAnimation*2,
radiusOfAnimation*2);
    Vector2 middle = new Vector2(radiusOfAnimation, radiusOfAnimation);
    for (int i = 0; i < radiusOfAnimation*2; i++)
    {
        for (int j = 0; j < radiusOfAnimation*2; j++)
        {
            Vector2 point = new Vector2(i, j);
            double distance = Vector2.Distance(point, middle);
            double difference = (radiusOfAnimation - distance) /
radiusOfAnimation;
            if (difference < 0) difference = 0;
            _terrHeights[i, j] = (float)(originalTerrainSectionHeight[i, j] *
(System.Math.Sin(Time.time + distance / 10) / 2f) * difference)*0.2f +
originalTerrainSectionHeight[i, j];
        }
    }
    _myTerrData.SetHeights(x, y, _terrHeights);
}
```

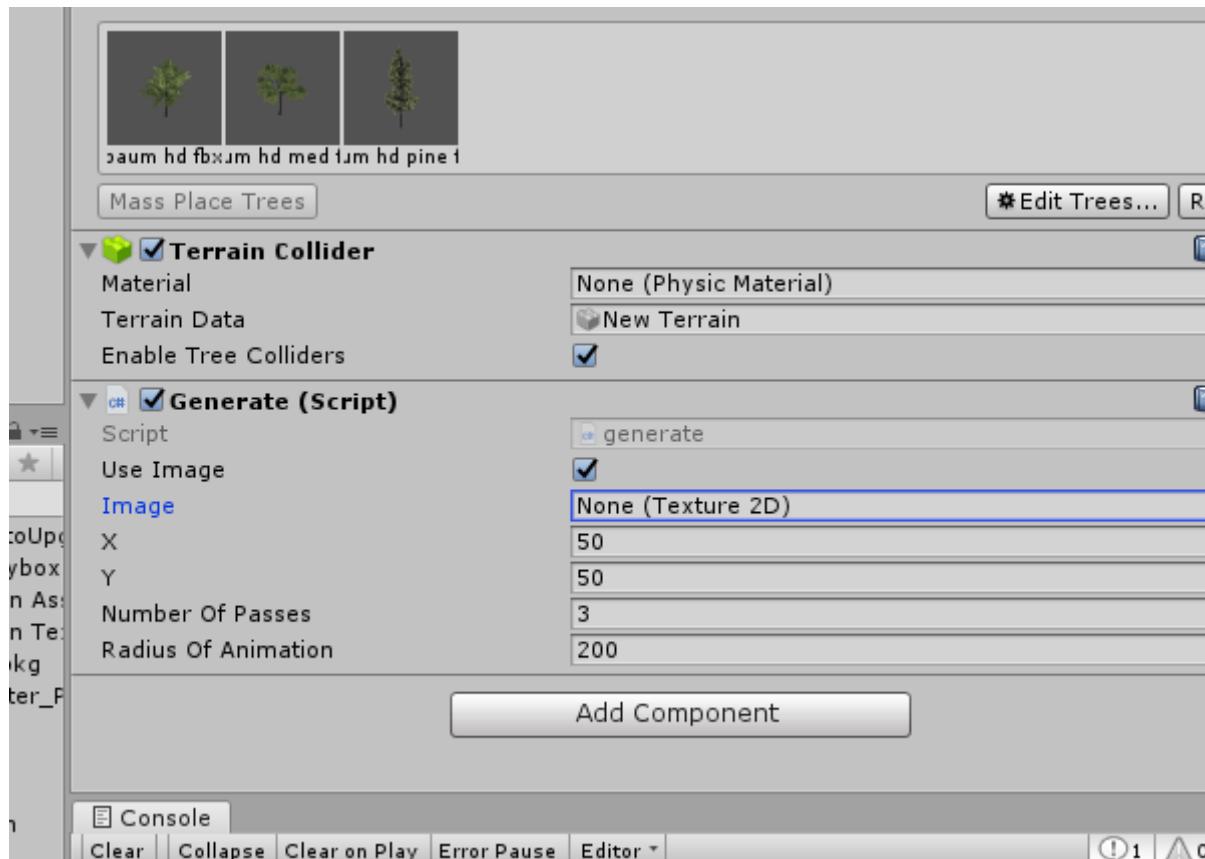
Krótki film prezentujący rezultat umieszczony w serwisie gyazo:

<https://i.gyazo.com/12636cacd9bb4244b6c1e0fea58095a7.mp4>



### 3. Wczytanie dowolnego napisu z pliku JPEG

Ostatnią częścią laboratorium było napisanie skryptu pozwalającego na wczytanie dowolnego obrazu i umiejscowieniu go na terenie (poprzez odpowiednie zmiany wysokości). Do obecnego skryptu dodano zmienną pozwalającą na wybór metody generacji terenu (Use Image) oraz umożliwiającą wybór obrazu (Image).



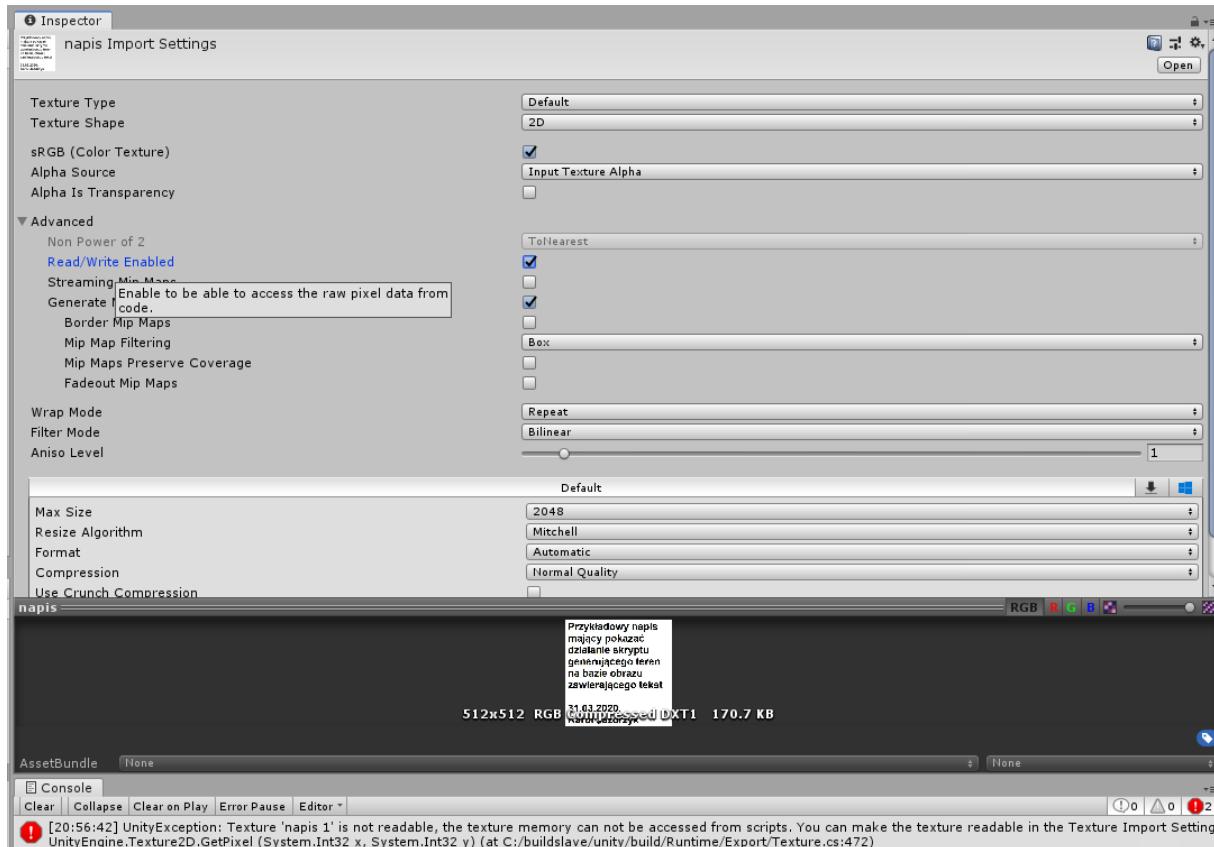
Napisana metoda prezentuje się następująco:

```
private void GenerateFromImage()
{
    _terrHeights = _myTerrData.GetHeights(0, 0, _xRes, _yRes);

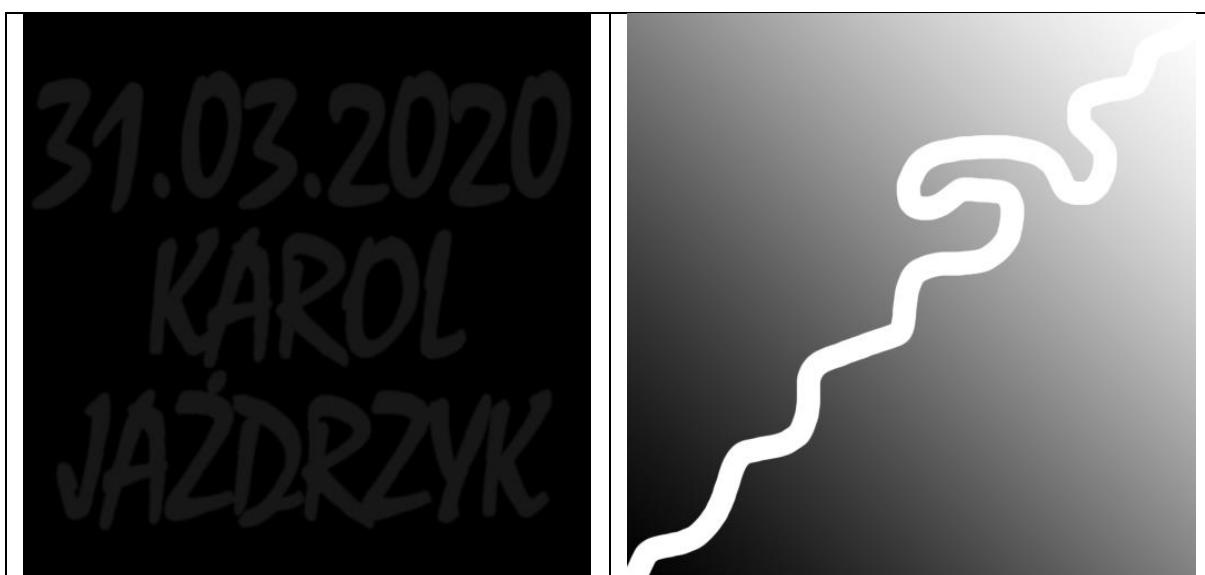
    for (int i = 0; i < _xRes; i++)
    {
        for (int j = 0; j < _yRes; j++)
        {
            Color pixel = image.GetPixel((int)((Convert.ToSingle(j)
                / Convert.ToSingle(_xRes)) * image.width),
                (int)((Convert.ToSingle(i) / Convert.ToSingle(_yRes))
                * image.height));
            _terrHeights[i, j] = (pixel.r + pixel.g + pixel.b)/3.0f;
        }
    }
    _myTerrData.SetHeights(0, 0, _terrHeights);
}
```

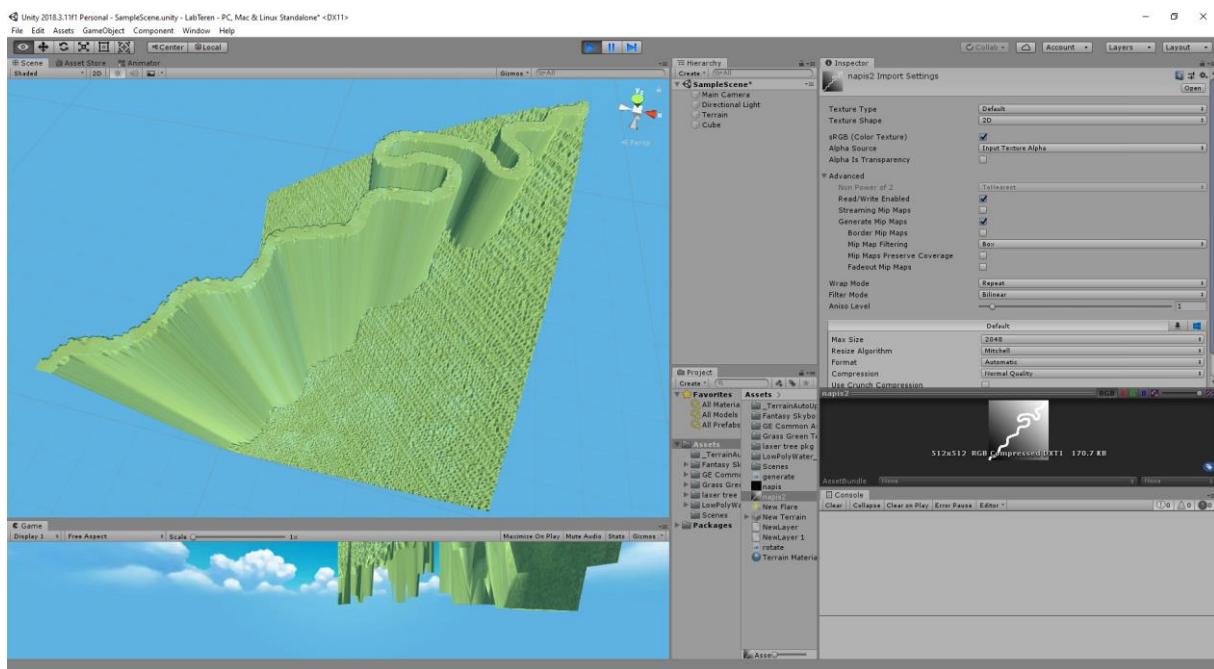
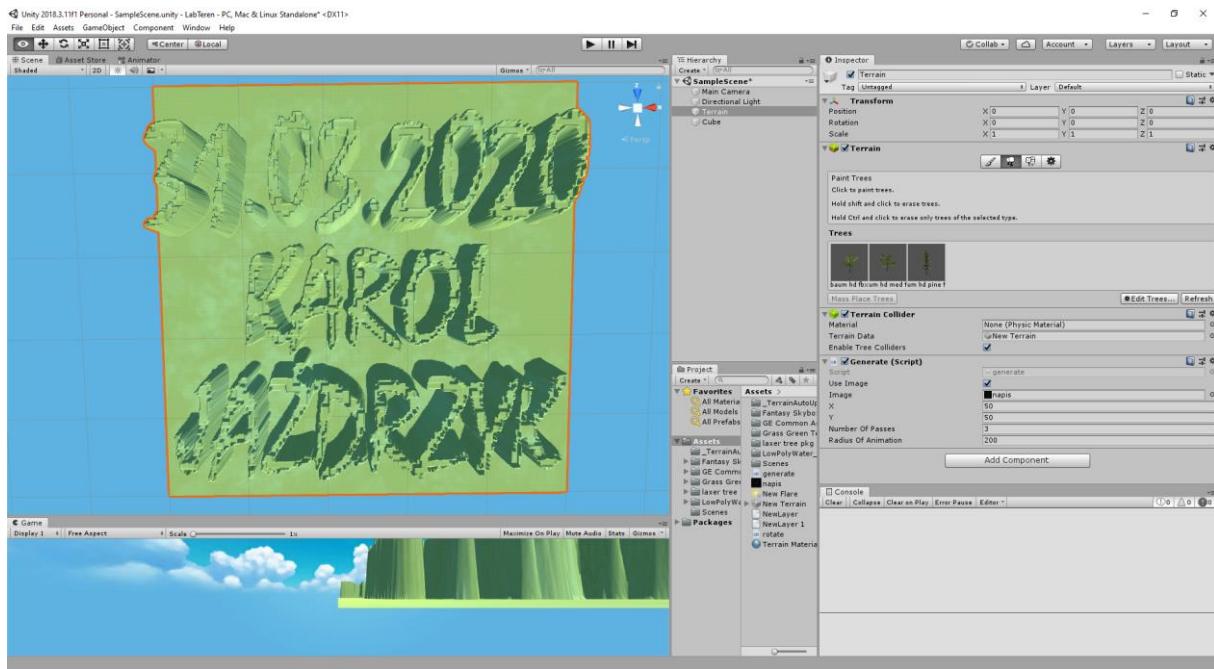
W przypadku gdy zmienna `useImage` jest prawdą, pomijana jest początkowa generacja na podstawie szumu oraz animacja zdefiniowana w poprzednim punkcie. Dla każdego wierzchołka w terenie przypisywana jest odpowiadająca wartość z rozciagniętego obrazka (skalowanie liniowe), a następnie jest przypisywana średnia wartość trzech składowych kolorów obrazu jako wysokość wierzchołka.

Aby można było odczytać wybraną teksturę, po zainportowaniu zasobu należało zaznaczyć opcję *Read/Write Enabled* w inspektorze obrazu:



Testowe obrazy:





Obrona metoda nie jest idealna, ale spełnia swoje zadanie. Wykorzystywany jest pełen zakres wartości wysokości od 0 do 1. W terenie występują drobne skoki w wysokości prawdopodobnie spowodowane szumem wprowadzanym przez format JPEG oraz próbkowaniem przeskalowanego obrazu. Lepsze rezultaty można by uzyskać stosując rozmycie po przypisaniu wysokości w celu usunięcia niedoskonałości lub innym algorytm usuwania szumów.

Link do kodu źródłowego w serwisie Github:  
<https://github.com/karo1404/ZTPGK-Teren>