Proceduralne generowanie terenu 3D z zastosowaniem silnika Unity

Autor: Cezary Muszyński

Cel projektu

Celem projektu jest stworzenie narzędzia do proceduralnej generacji terenu 3D w silniku gier Unity. Przykładowym sposobem wyko- rzystania takiego narzędzia jest tworze- nie gier. Używanie proceduralnej generacji w czasie produkcji gier pozwala znacząco przyspieszyć ten proces, jak również umożliwia tworzenie gier z nieograniczonym światem generowanym w czasie działania aplikacji.

Wykorzystane technologie

Silnik Unity
- Pakiet Lightweight RP
- Pakiet Shader Graph
- Standard Assets (for Unity 2018.4)

C#
- Biblioteka UnityEngine

Python
- Matplotlib

Wymagania niefunkcjonalne

- Celem projektu jest stworzenie narzędzia do generacji terenu (generatora terenu) działającego w środowisku Unity w wersji 2019.4.14f1 lub nowszej.

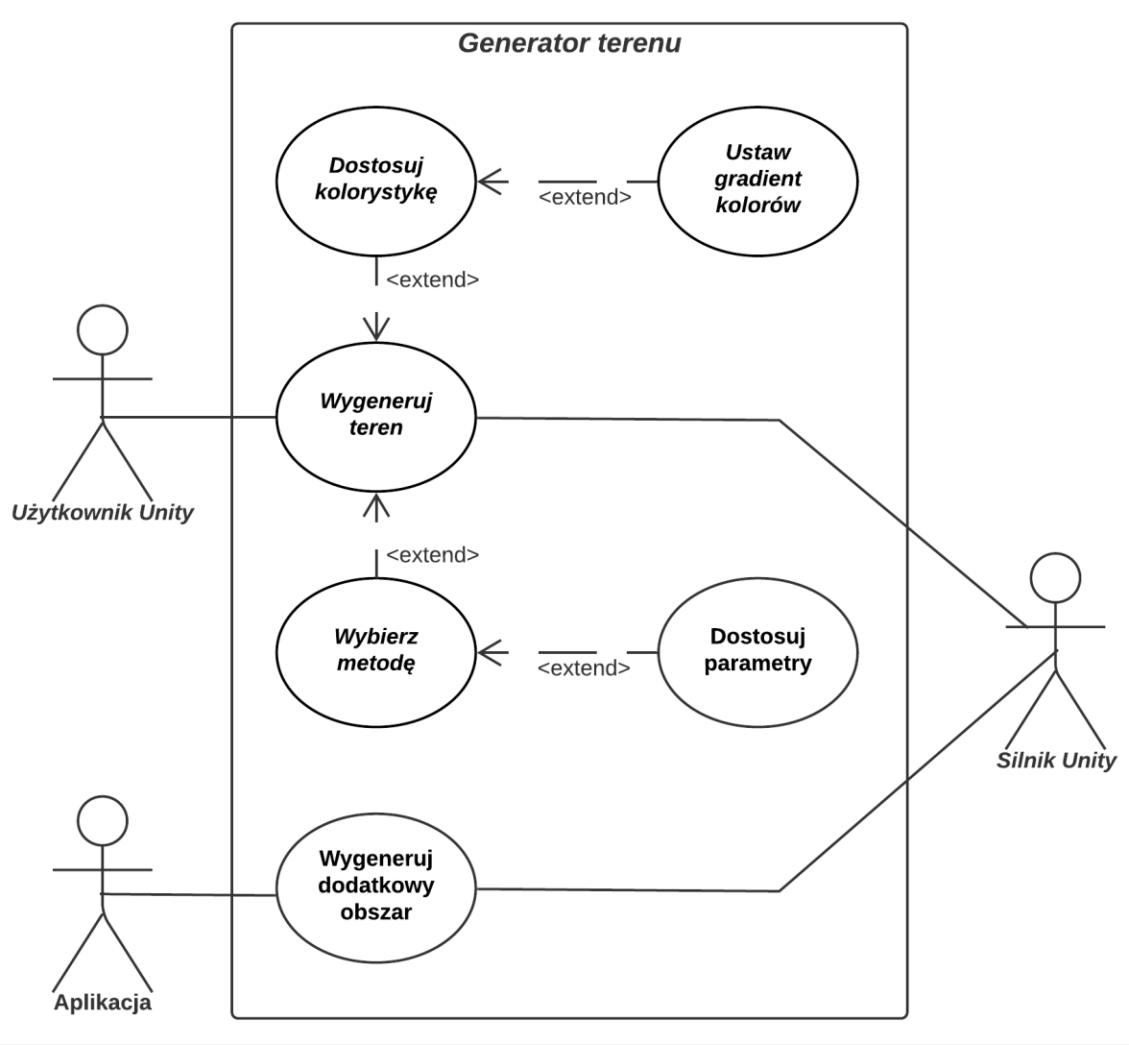
- Pożądana jest możliwość importu generatora do projektu rozwijanego w środowisku Unity niezależnie od ustawień i stopnia rozwoju projektu.

- Generator nie powinien wpływać na wymagania sprzętowe silnika Unity.

Zdecydowano, że najlepszym rozwiązaniem w celu spełnienia przedstawionych powyżej wymagań jest zaprojektowanie generatora jako tak zwany **prefab** z przyłączanym do niego skryptem. Prefab to obiekt silnika Unity zapisany wraz ze wszystkimi ustawieniami. Można go przenosić między projektami.

Wymagania funkcjonalne

- funkcja generacji terenu
  - funkcja zmiany metody generacji
    - funkcja zmiany parametrów terenu
  - funkcja dostosowania kolorystyki terenu
    - funkcja korzystania z gradientu kolorów
- funkcja generacji dodatkowej powierzchni terenu w czasie działania aplikacji

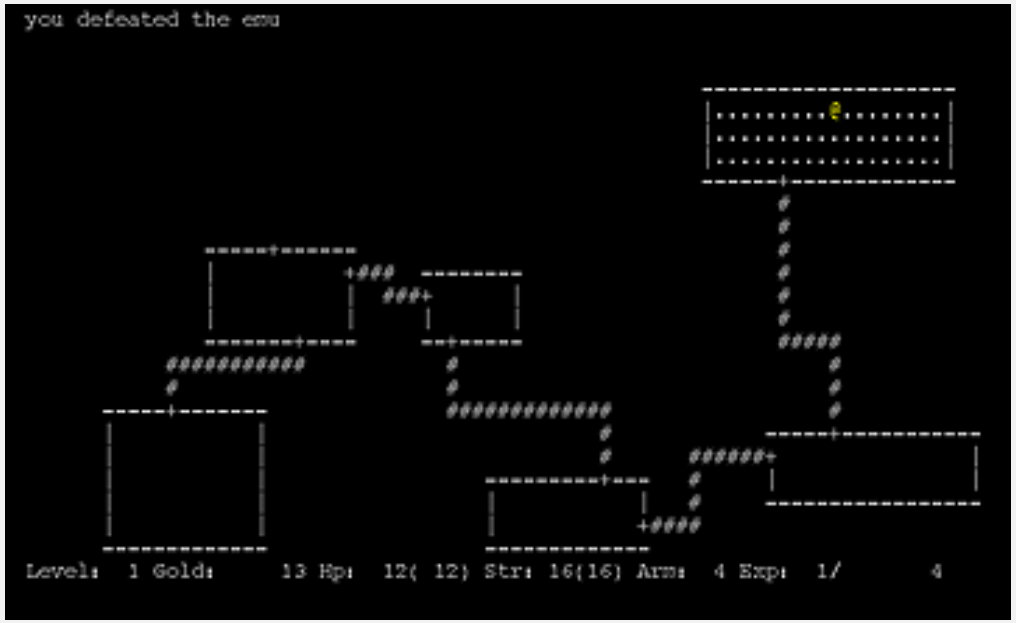


Rysunek 1: Diagram przypadków użycia, źródło własne.

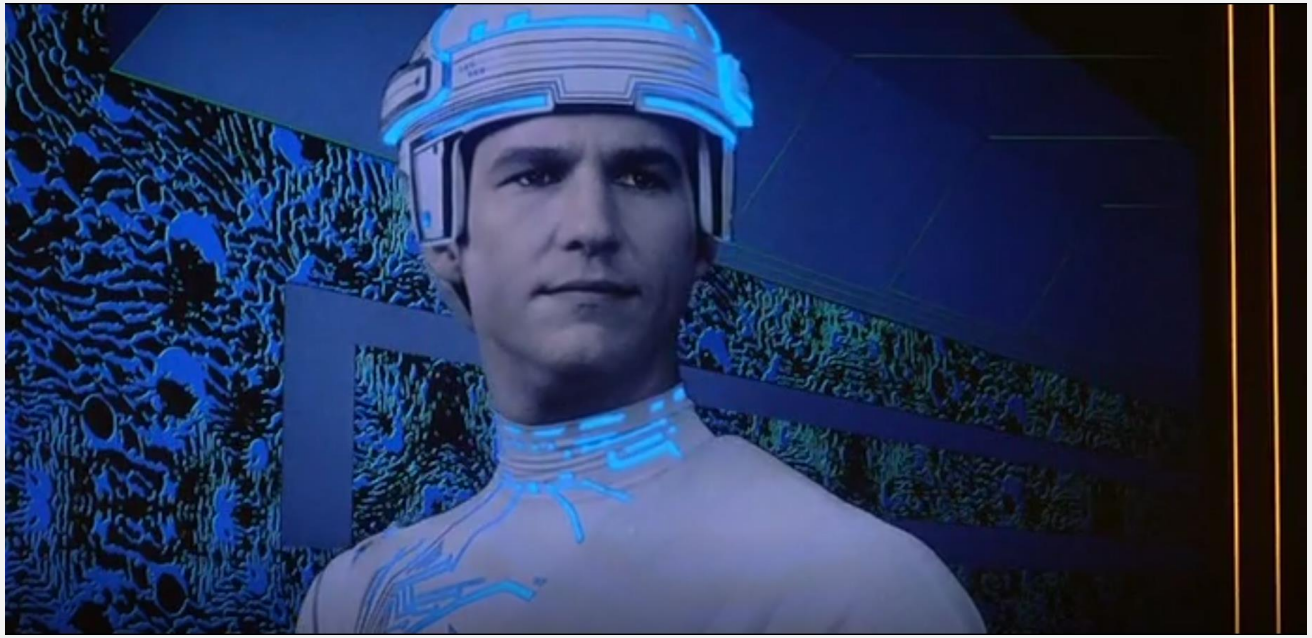
Analiza tematu

Temat projektu dotyczy proceduralnej generacji terenu, czyli zagadnienia z dziedziny proceduralnej generacji zawartości (ang. procedural content generation) określanej zazwyczaj (głównie w kontekście produkcji gier wideo) skrótem **PCG**. Wykorzystanie tej dziedziny w produkcji gier, filmów oraz dzieł pokrewnych sięga początków lat osiemdziesiątych, a obecnie jest powszechne.

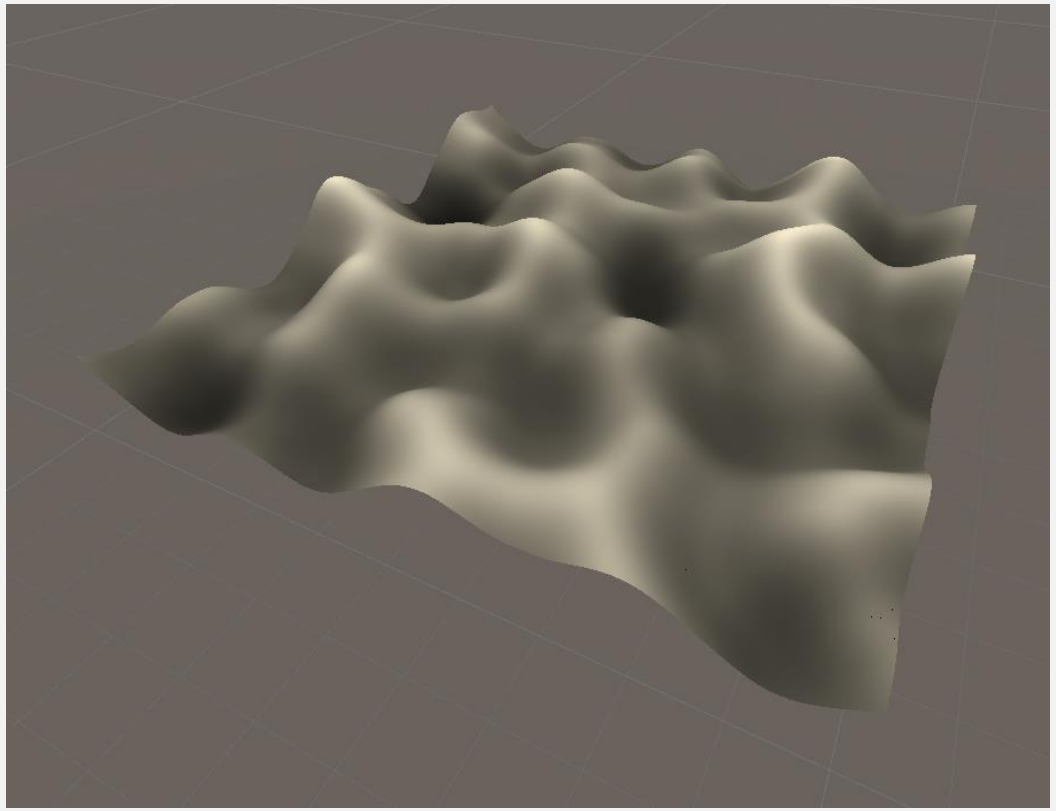
Do dzieł, które najbardziej przyczyniły się do popularyzacji PCG, można zaliczyć na przykład grę *Rogue* (1980), od której pochodzi nazwa gatunku roguelike. Przełomowe znaczenie miał również film *Tron* (1982). Podczas pracy nad nim Ken Perlin stworzył algorytm nazywany **Szumem Perlina**.



Rysunek 2: Zrzut ekranu przedstawiający proceduralnie wygenerowany poziom w grze Rogue, źródło: https://web.archive.org/web/20120815191124/http://www.edge-online.com/features/making-rogue



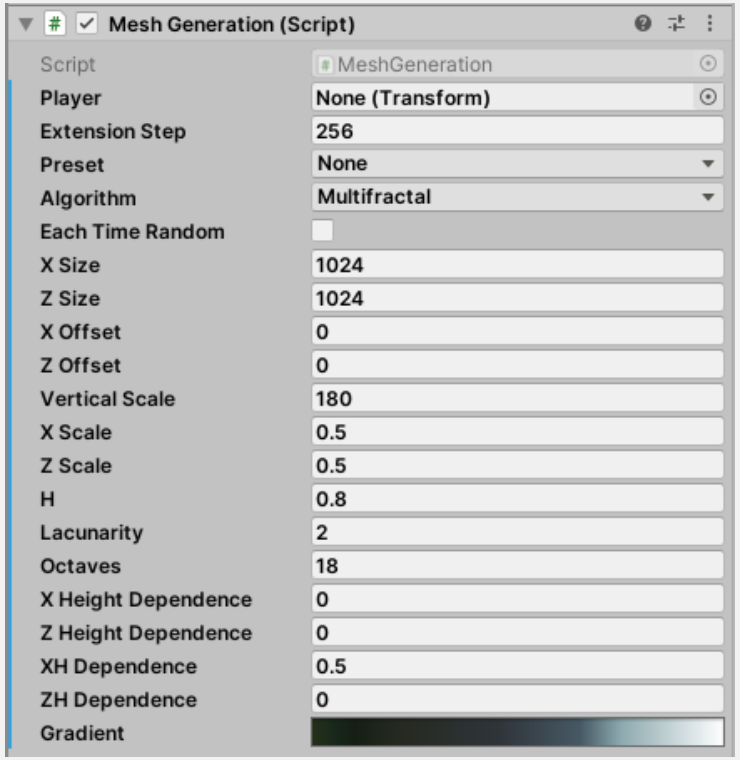
Rysunek 3: Zrzuty ekranu z filmu *Tron*, źródło własne.



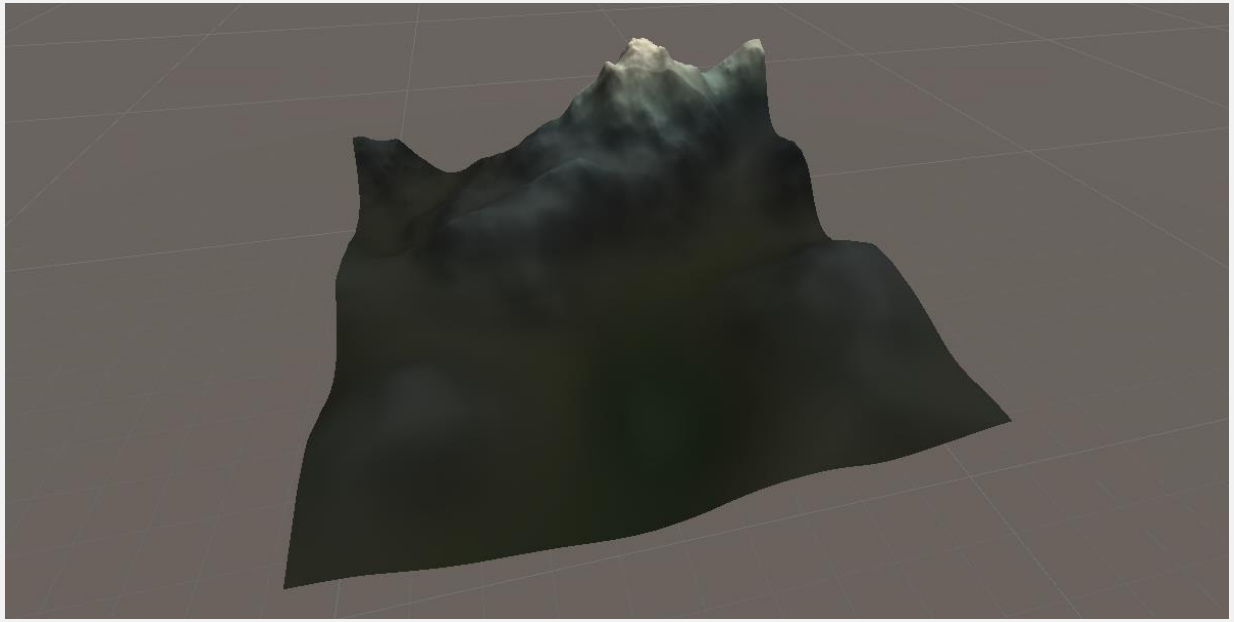
Rysunek 4: Szum Perlina przedstawiony jako teren 3D w silniku Unity, źródło własne.

Specyfikacja zewnętrzna

Generator terenu jest obsługiwany poprzez zmianę wartości parametrów skryptu *Mesh Generation* widocznego na ilustracji 5. Po wypełnieniu wszystkich pól należy uruchomić aplikację, aby zobaczyć uzyskany efekt. Wartości parametrów można zmieniać również w czasie działania aplikacji.



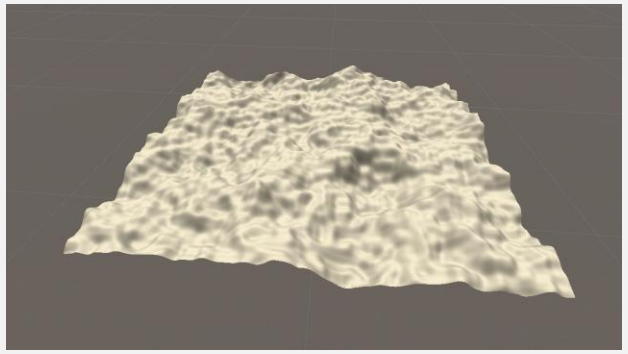
Rysunek 5: Widok skryptu *Mesh Generation* w oknie *Inspector* prefaba *TerrainGenerator*, źródło własne.



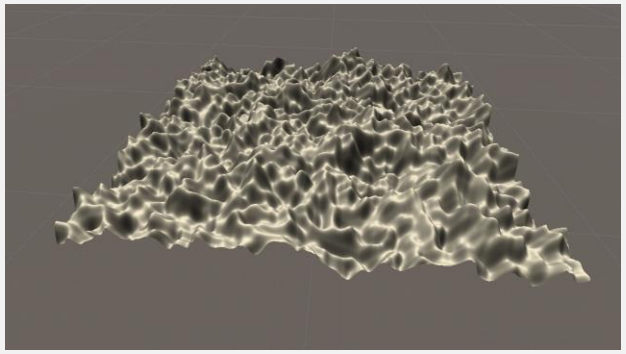
Rysunek 6: Teren wygenerowany według parametrów przedstawionych na rysunku 5., źródło własne.

Użytkownik stworzonego narzędzia ma do wyboru kilka algorytmów generacji terenu. W pracy największa uwaga została poświęcona metodom wykorzystującym fraktale, ponieważ pozwalały one na generowanie najciekawszych efektów.

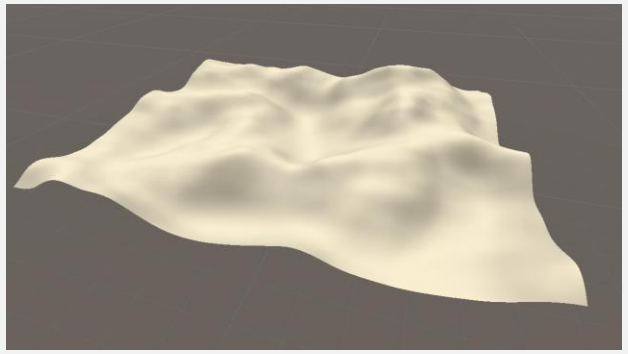
W przypadku większości parametrów dostępnych w skrypcie, ich wpływ na kształt terenu jest łatwy do przewidzenia, jednak część z nich służy do określania właściwości wykorzystanych fraktali. Dotyczy to na przykład parametrów *H*, *Lacunarity* i *Octaves*, które w języku polskim można określić odpowiednio jako **przyrost fraktalny**, **lakunarność** i **liczbę oktaw**. Ich wpływ na teren został zilustrowany przykładami na rysunkach 7. i 8.



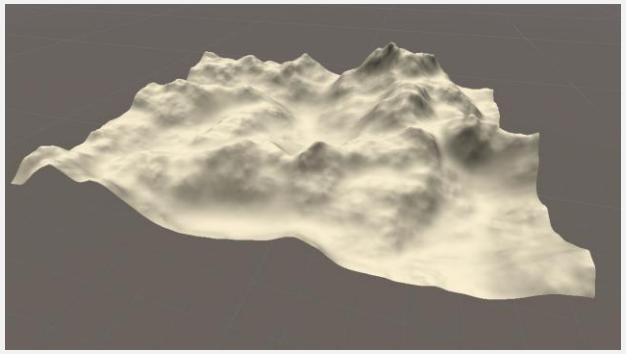
(a) *Fractional Brownian Motion*, *H* = 0.9



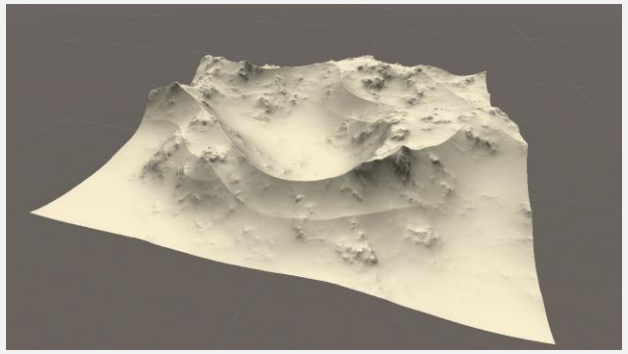
(b) *Fractional Brownian Motion*, *H* = 0.3



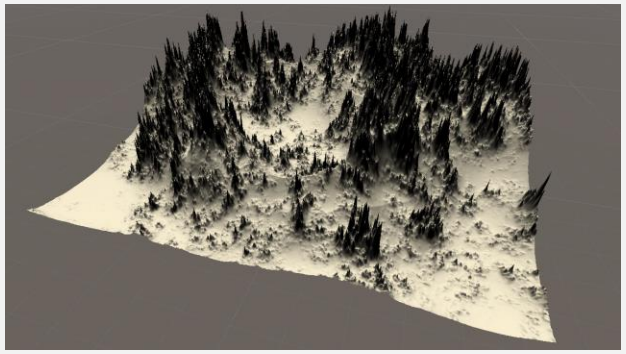
(c) *Multifractal*, *H* = 0.9



(d) *Multifractal*, *H* = 0.3

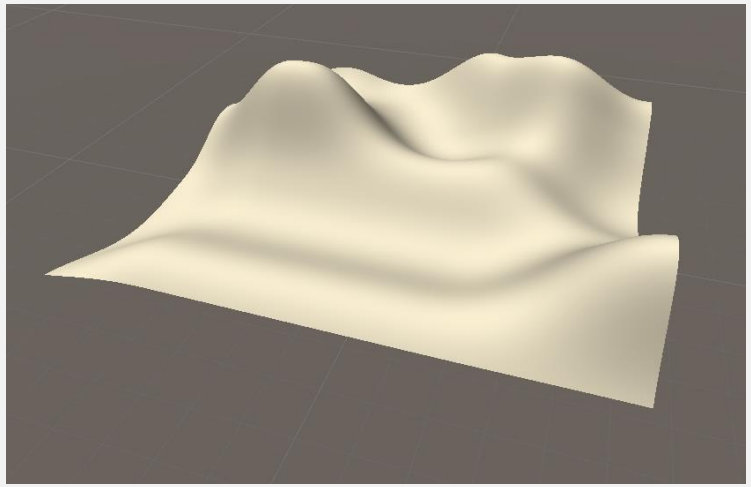


(e) *Ridged Multifractal*, *H* = 0.9

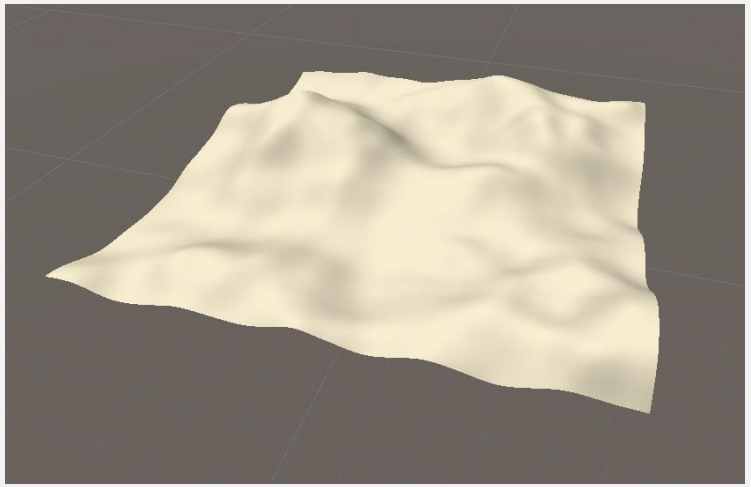


(f) *Ridged Multifractal*, *H* = 0.3

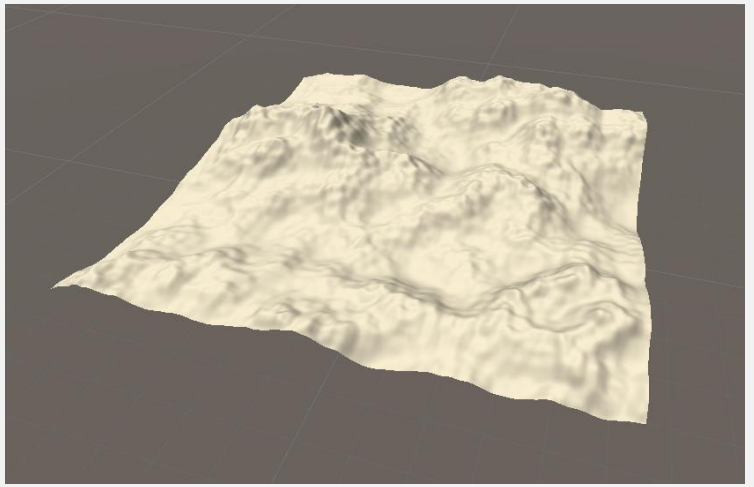
Rysunek 7: Wygląd generowanego terenu w zależności od wybranego algorytmu i wartości parametru *H*, źródło własne



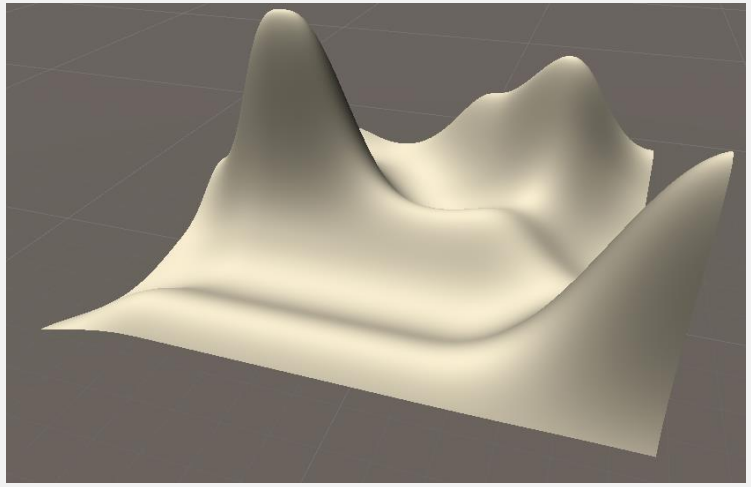
(a) *Lacunarity* = 1, *Octaves* = 3



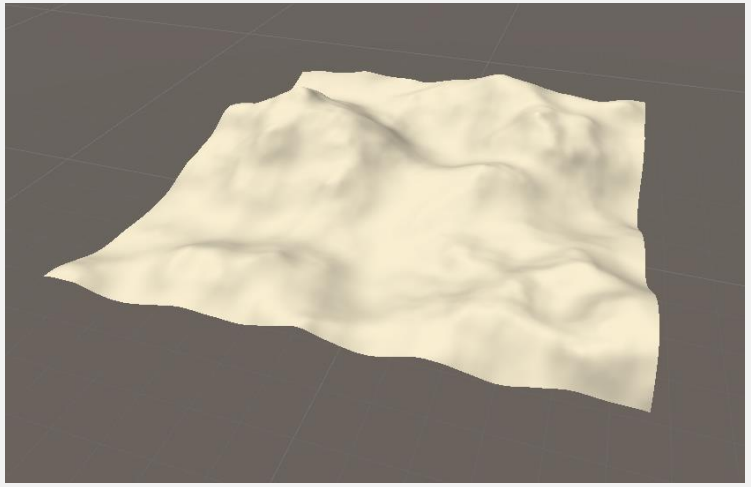
(b) *Lacunarity* = 2, *Octaves* = 3



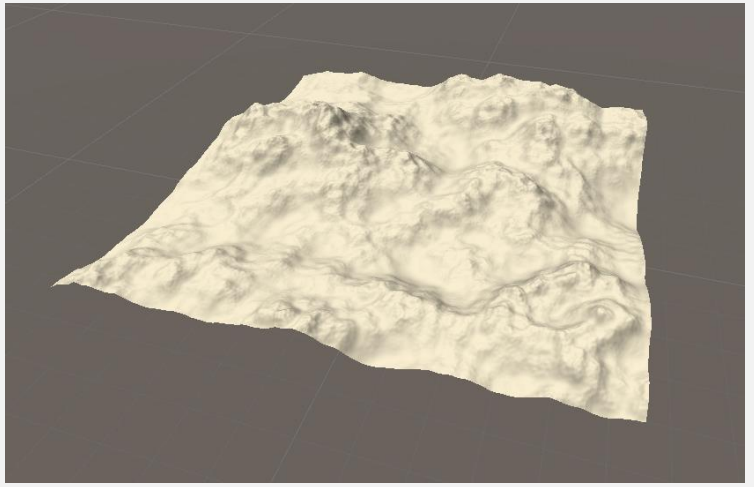
(c) *Lacunarity* = 4, *Octaves* = 3



(d) *Lacunarity* = 1, *Octaves* = 9



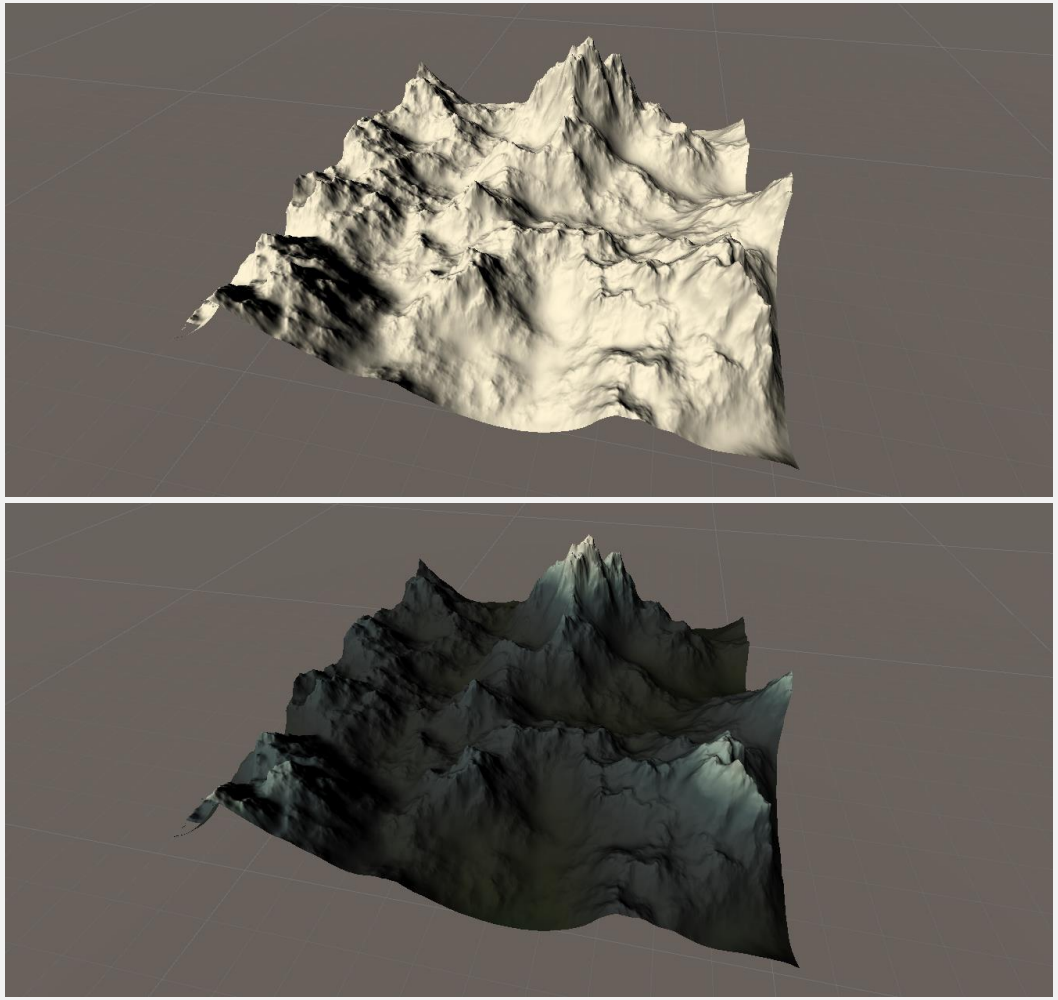
(e) *Lacunarity* = 2, *Octaves* = 9



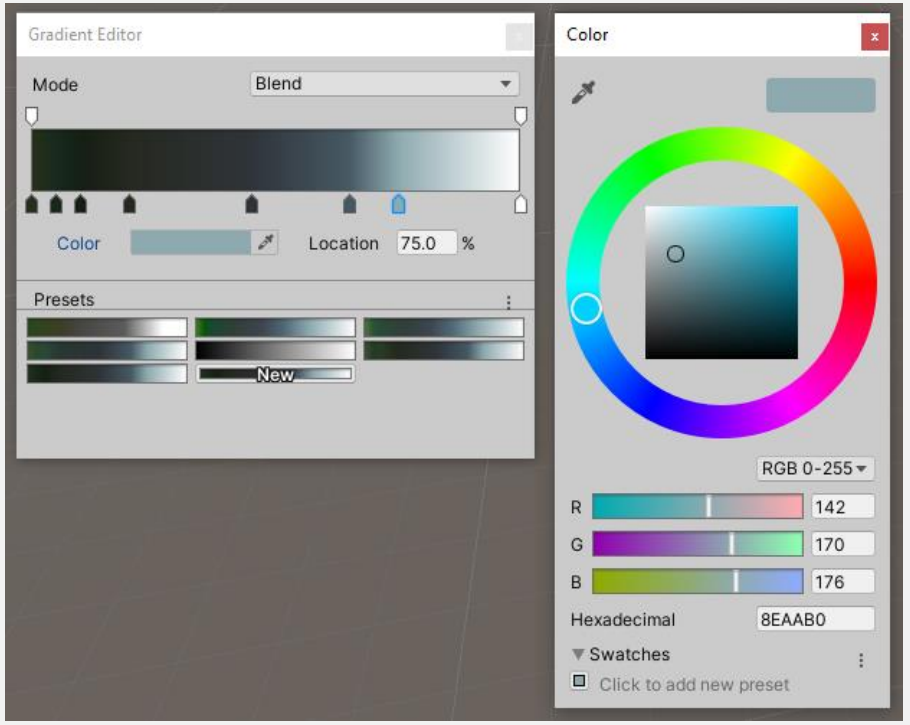
(f) *Lacunarity* = 4, *Octaves* = 9

Rysunek 8: Wygląd terenu generowanego przez algorytm *Multifractal* w zależności od lakunarności i liczby oktaw, źródło własne.

W projekcie zaimplementowano możliwość dostosowania koloru terenu, w tym opcję wykorzystania gradientu wyświetlanego w taki sposób, że kolor zależy od wysokości terenu.



Rysunek 9: Efekt generacji terenu przy użyciu domyślnego materiału oraz z wykorzystaniem gradientu, źródło własne.

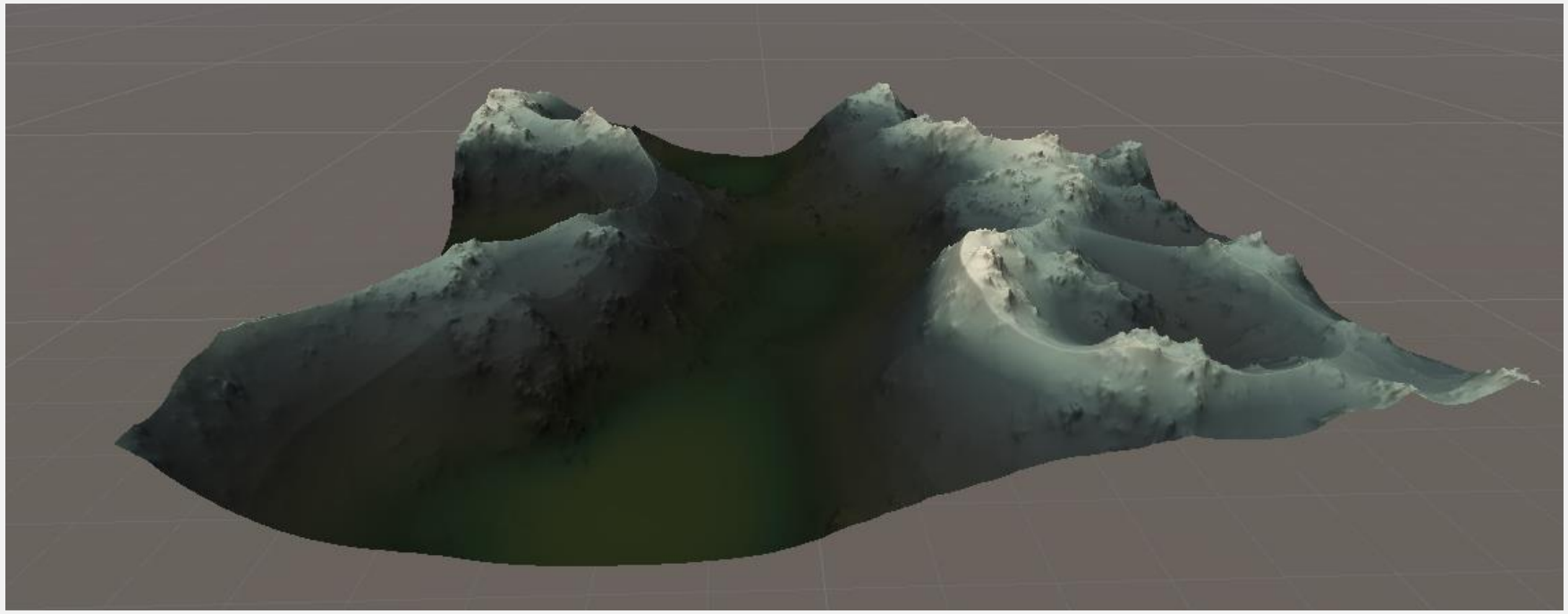


Rysunek 10: Edytor gradientu, źródło własne.

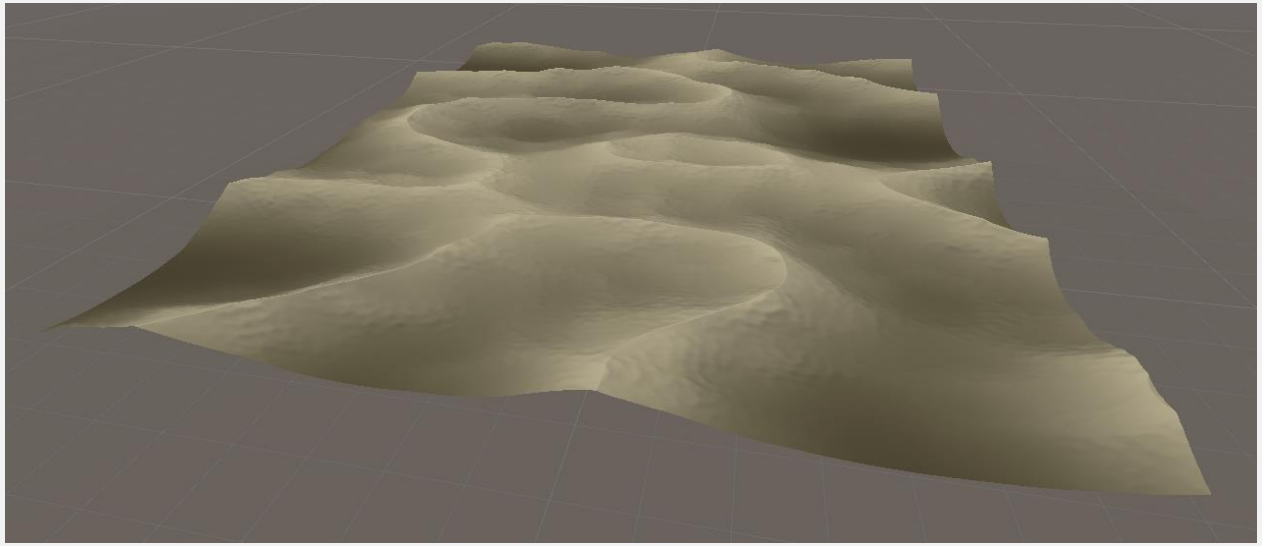
Specyfikacja wewnętrzna

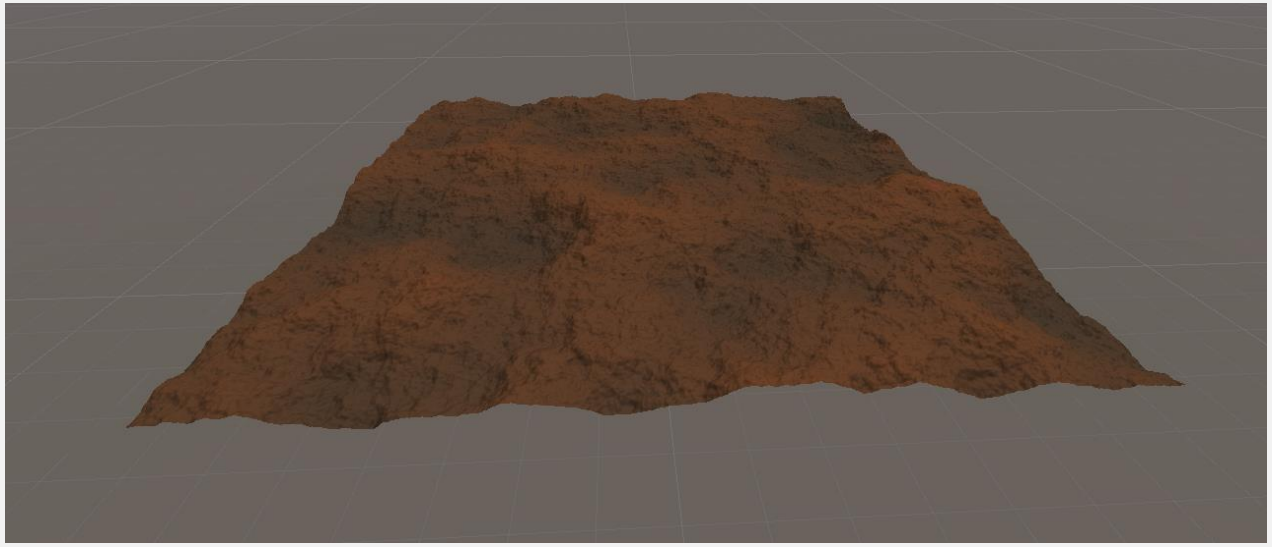
Ogólna zasada działania generatora terenu zaprojektowanego w ramach niniejszego projektu jest następująca.

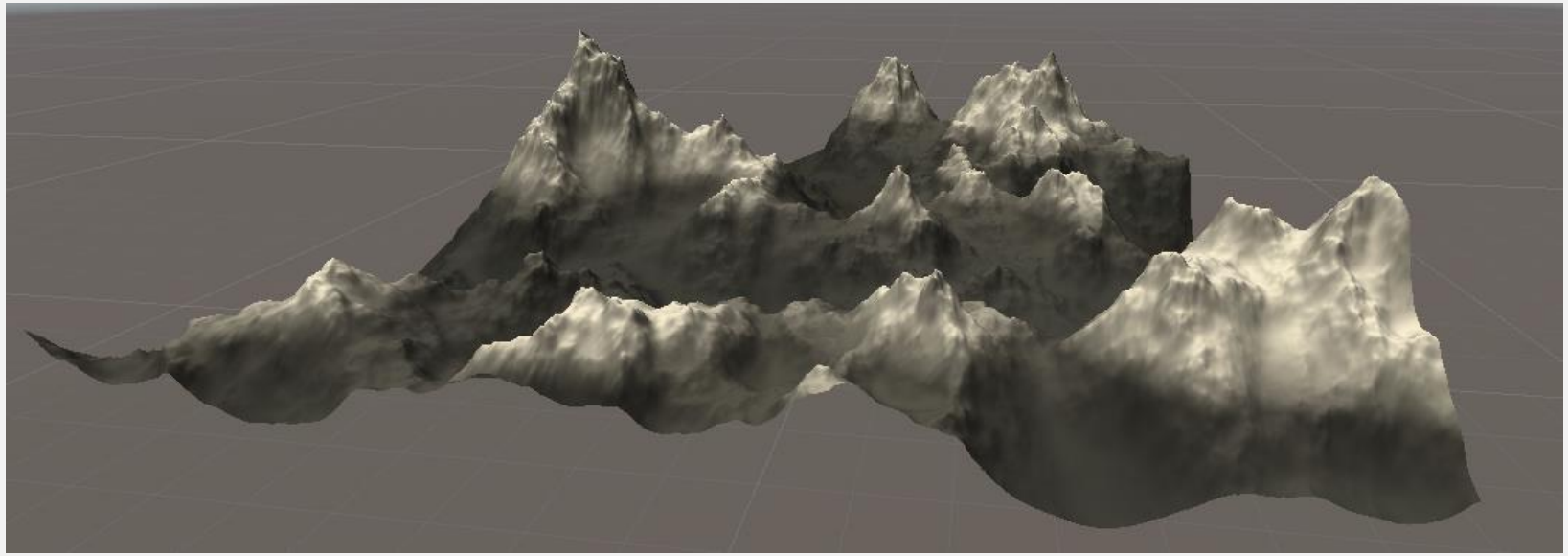
- W pierwszym etapie użytkownik wprowadza dane do skryptu.
- W drugim etapie na podstawie wprowadzonych danych generowana jest mapa wysokości.
- W trzecim etapie mapa wysokości zostaje wykorzystana, aby utworzyć tak zwany *mesh,* czyli siatkę wierzchołków połączonych ze sobą tak, aby formować obiekty 3D.



Rysunek 11:
Teren generowany z wykorzystaniem funkcji *RidgedMultifractal(x,z)*,
źródło własne.







Rysunek 12:
Inne przykłady terenu stworzonego za pomocą generatora, źródło własne.

Podsumowanie i wnioski

Projekt spełnia zdefiniowane wymagania, zarówno funkcjonalne jak i niefunkcjonalne. Generator pozwala użytkownikowi na wybór jednego z pięciu algorytmów generacji mapy wysokości, spośród których dwa mogą posłużyć do generacji terenu wyglądającego atrakcyjnie i mogącego kojarzyć się z naturalnymi krajobrazami. Pozostałe algorytmy mogą służyć do przeprowadzania eksperymentów lub do generacji terenu wyglądającego abstrakcyjnie. W przypadku każdego z algorytmów, właściwości generowanego terenu (w tym kolorystykę) można dostosować za pomocą zestawu parametrów.

W ramach dalszego rozwoju projektu bardzo korzystne byłoby na przykład poprawienie optymalizacji. Jednym ze sposobów, by tego dokonać jest zastosowanie techniki znanej pod nazwą *Level of detail*. Pozowala ona na zmniejszenie liczby przetwarzanych wierzchołków w zależności od odległości, w jakiej znajduje się wyświetlany obiekt.

Dziękuję za uwagę!

Cezary Muszyński