

# Landscape Generator

## Inżyniera Oprogramowania

Artur Bednarczyk, Dawid Grajewski, Tomasz Januszek  
Politechnika Śląska  
Wydział Matematyki Stosowanej  
Informatyka, semestr V

9 stycznia 2019

# Spis treści

<b>1</b>	<b>O projekcie</b>	<b>3</b>
1.1	Zespół . . . . .	3
1.2	Temat . . . . .	3
<b>2</b>	<b>Projekt</b>	<b>3</b>
2.1	Plany i pomysły . . . . .	3
2.2	UI/UX . . . . .	3
2.2.1	Zawartość . . . . .	3
2.2.2	Projekty UI . . . . .	5
<b>3</b>	<b>Teoria</b>	<b>6</b>
3.1	Losowość . . . . .	6
3.2	Algorytmy . . . . .	6
3.2.1	Szum Perlina . . . . .	6
<b>4</b>	<b>Narzędzia</b>	<b>6</b>
4.1	Kontrola wersji . . . . .	6
4.2	Zarządzanie zespołem . . . . .	7
4.3	Środowisko . . . . .	7
<b>5</b>	<b>Aplikacja</b>	<b>7</b>
5.1	Architektura . . . . .	7
5.1.1	Silnik graficzny - InsightEngine . . . . .	7
5.1.2	Wygenerowany teren - TerrainGenerator . . . . .	8
5.1.3	Generator terenu - PerlinNoise . . . . .	8
5.1.4	Interfejs użytkownika - UI . . . . .	8
5.1.5	Połączenie . . . . .	8
5.2	Struktury danych . . . . .	9
5.3	Schemat graficzny struktury systemu . . . . .	9
<b>6</b>	<b>API</b>	<b>9</b>
6.1	PerlinNoise . . . . .	9
6.2	InsightEngine . . . . .	10

# 1 O projekcie

## 1.1 Zespół

Osoba	Główna odpowiedzialność
Artur Bednarczyk	Algorytm generujący kształt terenu, organizacja, dokumentacja
Dawid Grajewski	Silnik wyświetlający teren
Tomasz Januszek	UI aplikacji, algorytm generujący kształt terenu

## 1.2 Temat

**Generowanie realistycznych krajobrazów 3D** Generowanie w języku wysokiego poziomu (nie w generatorach typu Unity) losowych krajobrazów z uwzględnieniem zadanych parametrów: stromizny terenu, poziomu wody, kolorów na danej wysokości lub obszarzem wizualizacja i symulacja prze-mieszczania kamery.

Bonus: dodanie roślinności (drzewa, krzewy - co najmniej 3 rodzaje) o zadanej częstości i miejscu występowania.

# 2 Projekt

## 2.1 Plany i pomysły

Zgodnie z założeniami projektu, głównym naszym celem jest wygenerowanie losowego krajobrazu, który będzie realistyczny, przy czym nie wykorzystamy gotowych silników typu Unity. Plan jest taki, aby użytkownik mógł podać parametry, zgodnie z którymi zostanie wygenerowany krajobraz oraz miał możliwość zapisania i odczytania wybranego krajobrazu. Chcemy również dodać możliwość wyświetlania dodatkowych elementów, takich jak drzewa, krzewy.

## 2.2 UI/UX

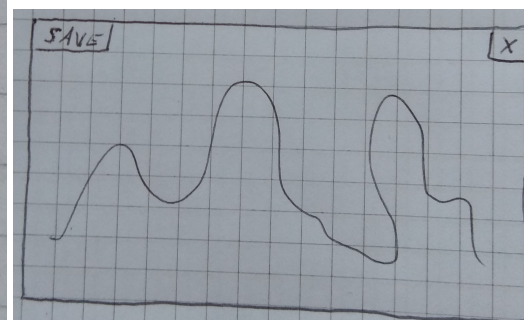
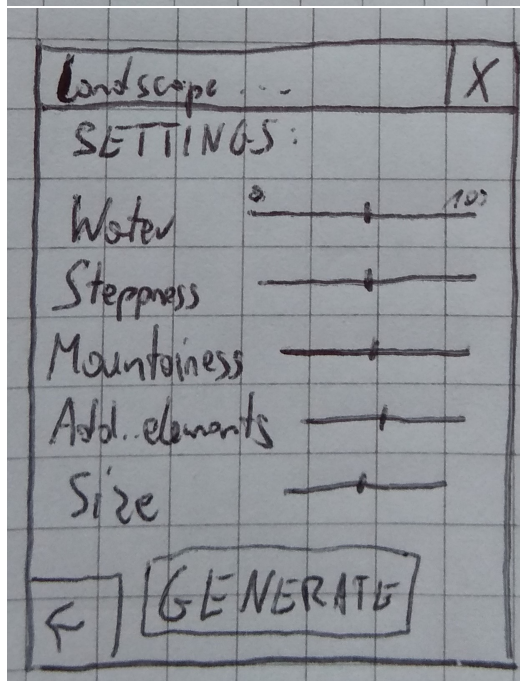
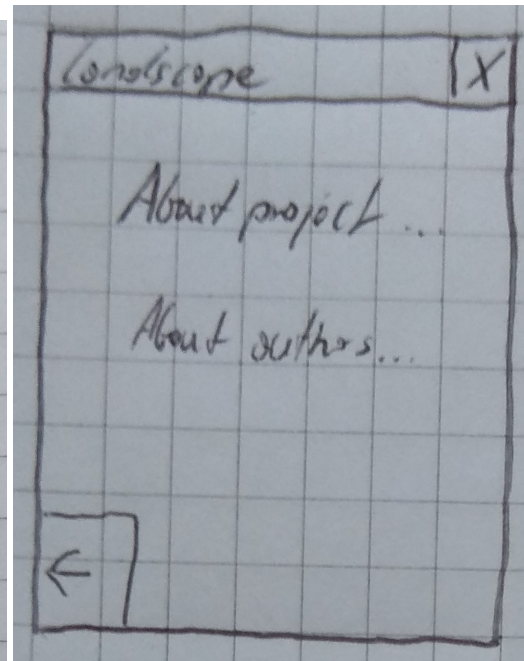
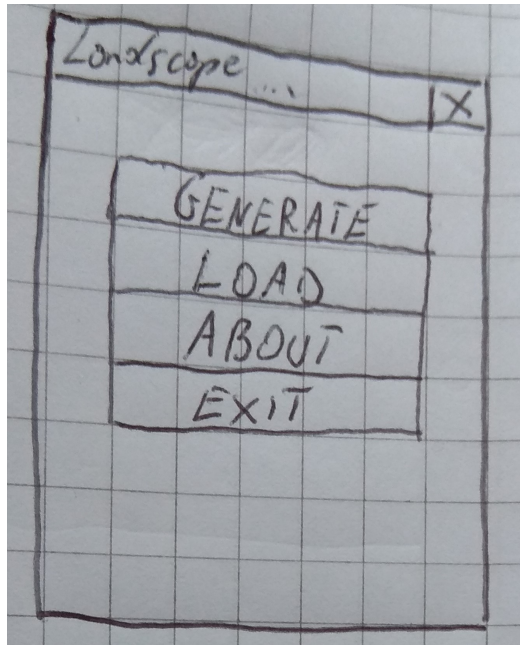
### 2.2.1 Zawartość

UI programu będzie złożone z kilku elementów:

- Menu - pozwoli na przejście do ustawień generowania, informacji o aplikacji oraz do wczytania wcześniej zapisanego krajobrazu.
- Ustawienia generowania - pozwoli na ustawienie konkretnych parametrów, zgodnie z którymi zostanie wygenerowany krajobraz. Parametry te to:

- Stromizna terenu
  - Poziom wody
  - Górzystość
  - Gęstość dodatkowych elementów
  - Położenie dodatkowych elementów
  - Rozmiar krajobrazu
- Informacje - o aplikacji, autorach
- Wygenerowany krajobraz - tutaj użytkownik może się poruszać po obszarze, który został wygenerowany oraz go zapisać.

## 2.2.2 Projekty UI



## 3 Teoria

### 3.1 Losowość

Celem projektu jest wygenerowanie losowego krajobrazu, więc sama losowość jest bardzo ważna. Każdy kolejny krajobraz powinien być inny, a prawdopodobieństwo wystąpienia dwóch podobnych obrazów bardzo niskie. Takie efekty możemy uzyskać dzięki algorytmowi jakim jest „Szum Perlina”

### 3.2 Algorytmy

#### 3.2.1 Szum Perlina

Bazę dla naszego sposobu generowania danych potrzebnych do wyświetlenia zróżnicowanego terenu stanowi szum Perlina. Jest to jeden z typów szumu gradientowego utworzony przez Kena Perlina już w 1983 roku dla potrzeby tworzenia realistycznych grafik komputerowych. Algorytm składa się z trzech kroków

- Pierwszym krokiem jest zdefiniowanie wielowymiarowej siatki jednostkowych wektorów rozpatrywanego gradientu. W naszym przypadku są to wartości losowe z zakresu  $(0, 1)$  liczb rzeczywistych. Dla jednowymiarowego przypadku byłyby dostępne jedynie wartości  $-1$  albo  $1$
- Kolejno iteruje się po podawanych punktach. Punkt wpada do pewnej komórki wygenerowanej siatki. Następnie wyliczany jest iloczyn skalarny między punktem a wektorem każdego z rogów komórki (a więc ich odległość), po czym zapisane zostają w pamięci.
- Przeprowadzona zostaje interpolacja dla każdej pary punktów z uwzględnieniem funkcji wygładzającej.

Wynikowo otrzymujemy wielowymiarową macierz (lub tensor) zawierający wartości zamknięte w pewnych granicach. Jest możliwe nakładanie na siebie wielu takich macierzy generowanych dla różnych częstotliwości siatki w celu uzyskania różnych ułożeń lub skupień wartości.

## 4 Narzędzia

### 4.1 Kontrola wersji

Do zarządzania kodem i wersjami projektu wykorzystujemy narzędzie Git. Korzystamy z platformy GitHub jako repozytorium dostępnego online. Do-

bór narzędzi służących do korzystania z repozytorium to sprawa indywidualna każdego członka zespołu, ponieważ nie ma ona wpływu na sam projekt.

## 4.2 Zarządzanie zespołem

Trello - Kanban Board - to tutaj rozpisujemy zadania i przydzielamy je sobie, określamy również terminy i planujemy.

## 4.3 Środowisko

Visual Studio 2015

Do wyświetlenia terenu wykorzystujemy zestaw funkcji API wspomagający generowanie grafiki – DirectX wraz z DirectX Software Development Kit używany z C#.

# 5 Aplikacja

## 5.1 Architektura

### 5.1.1 Silnik graficzny - InsightEngine

Utworzony silnik graficzny jest wzorowany na istniejących już popularnych silnikach do gier takich jak Unity, Cry czy Unreal Engine. Opiera się on na trzech głównych składowych:

- scena - obiekt nadrzędny, to w nim rozgrywa się akcja. Zawiera kolekcje encji.
- encja - umieszczona w scenie. Jest to pojedynczy agent w scenie, np. kamera, teren czy postać. Zawiera kolekcje komponentów.
- komponent - to składowa encji - odpowiada za część logiki encji (przykładowo, encja “player” może mieć komponenty takie jak “moveComponent” czy “shootComponent”

Silnik jest zbudowany zgodnie z wzrocom kompozytu - encja składa się z komponentów, ale pojedynczy komponent może zawierać kolejne, zagnieżdżone komponenty. Pozwala to na rekurencyjne wykonywanie metod w obiektach o tym samym interfejsie.

Z każdą kolejną wyświetlaną klatką scena wykonuje aktualizację wszystkich

swoich encji, które aktualizują swoje komponenty. Pozwala to m.in. na ruch kamery.

Specjalne komponenty typu “Renderer” klasy “ShapeRenderer” są odpowiedzialne za rysowanie na scenie obiektów 3D.

### **5.1.2 Wygenerowany teren - TerrainGenerator**

Forma odpowiedzialna za wyświetlenie efektu działania silnika graficznego. Forma wywoływana jest z zestawem argumentów odpowiedzialnym za parametry generowanego terenu lub ścieżkę do pliku z zapisanym terenem.

### **5.1.3 Generator terenu - PerlinNoise**

Klasa odpowiedzialna za generowanie kształtu terenu to PerlinNoise, implementuje interfejs INoiseGenerator. Ustawienia konfiguracyjne generatora znajdują się w folderze Config w plikach PerlinParameters oraz PerlinValueSetter. Aby skorzystać z generatora należy utworzyć jego obiekt, który w konstruktorze wymaga rozmiaru generowanego obszaru i liczby wymiarów, w których liczymy. Następnie należy wywołać metodę CalculateNoiseValue, która jako argumenty przyjmuje współrzędne punktu oraz rozmiar mapy.

### **5.1.4 Interfejs użytkownika - UI**

UI jest utworzone w projekcie WindowsForms. Zbudowane zgodnie z architekturą Model View Presenter.

- VIEW - jedna forma odpowiedzialna za przełączanie między UserControl oraz utworzenie połączenia z prezenterami. Każdy obiekt UserControl jest umieszczony w osobnym folderze, gdzie posiada swój interfejs.
- PRESENTER - każdy widok posiada tutaj swój odpowiedni folder, w którym znajduje się prezenter.
- MODEL - model posiadający dane ENUM.

### **5.1.5 Połączenie**

UI w momencie, gdy użytkownik chce wygenerować nowy lub wczytać z pliku teren, wywołuje formę TerrainGenerator, która wyświetli wynik pracy InsightEngine, który do generowania korzysta z PerlinNoise.



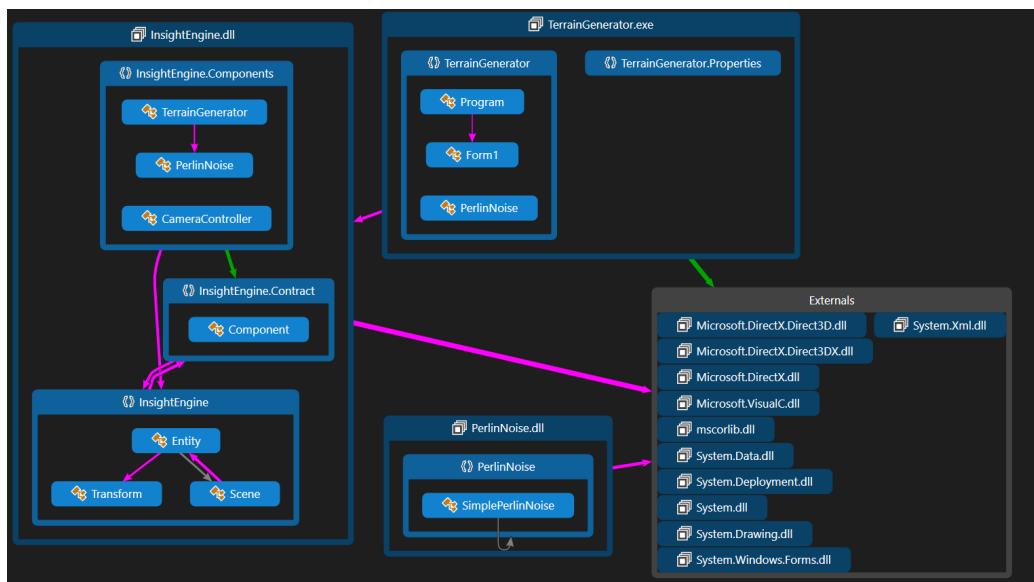
## 5.2 Struktury danych

Dane o wygenerowanym terenie będą przechowywane w pliku. Umożliwi nam to zapisanie i późniejsze odtworzenie terenu. Plik będzie przechowywał tablice odpowiadające za kształt terenu wraz z kolorami oraz drugą tablicę, która będzie przechowywać informacje na temat położenia dodatkowych elementów wraz z typem danego elementu. Przykładowy plik:

```
1 [{x:y:z:c};{x:y:z:c}]
2 [{x:y:z:t};{x:y:z:t}]
```

Gdzie x,y,z to współrzędne, c to kolor, a t to tag odpowiadający za typ modelu.

## 5.3 Schemat graficzny struktury systemu



## 6 API

### 6.1 PerlinNoise

Konstruktor: `PerlinNoise(int gradientSize, int numberOfDimensions)` - gradientSize to rozmiar generowanego terenu, liczba wymiarów. Dla generatora 3D liczba wymiarów równa 3.

Pobranie wartości: `CalculateNoiseValue(int x, int z, int Size)` - x,z to współrzędne, Size to rozmiar generowanego terenu.

## 6.2 InsightEngine