МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

Кваліфікаційна робота магістра

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ГЕНЕРАЦІЇ ПРИРОДНОГО ЛАНДШАФТУ НА ОСНОВІ ШУМУ ПЕРЛІНА

Здобувач освіти гр. ІН м. – 12ан Вячеслав ФОМЕНКО

Науковий керівник,
ст. викл., к.ф.-м.н. Оксана ШОВКОПЛЯС

В.о. завідувача кафедри
доцент, к.т.н. Ігор ШЕЛЕХОВ

Суми 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Затверджую_____
В.о. зав. кафедри Шелехов І. В.
«_____» _____2022 р.

ЗАВДАННЯ
до кваліфікаційної роботи

здобувача вищої освіти другого курсу магістратури, групи ІН м.-12ан спеціальності «122 – Комп'ютерні науки» денної форми навчання Фоменко Вячеслава Олександровича.

Тема: «Інформаційна технологія генерації природного ландшафту на основі шуму Перліна»

Затверджена наказом по СумДУ
№_____ від _____2022 р.

Зміст пояснювальної записки: 1) Огляд технологій розробки та існуючих рішень, що застосовуються для генерації ландшафту. 2) Формування мети та постановки задачі, визначення етапів реалізації проєкту. 3) Вибір технологій та інструментів, що необхідні для розробки 4) Розробка use-case, DTF діаграм 5) Розробка основної частини(генерації ландшафту 6) Тестування готового програмного продукту та аналіз результатів.

Дата видачі завдання «_____»_____ 2022 р.

Керівник роботи _____ О. А Шовкопляс

Завдання прийняв до виконання _____ В. О Фоменко

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Термін виконання проектуі (роботи)	Примітка
1.	Огляд технологій розробки та існуючих рішень, що застосовуються для генерації ландшафту.		
2.	Формування мети та постановки задачі, визначення етапів реалізації проекту.		
3.	Вибір технологій та інструментів, що необхідні для розробки		
4.	Розробка use-case, DTF діаграм		
5.	Оформлення пояснювальної записки до дипломної роботи		
6.	Тестування готового програмного продукту та аналіз результатів.		

Студент–дипломник _____
(підпис)

Керівник проєкту _____
(підпис)

РЕФЕРАТ

Записка: 70 стор., 23 рис., 1 додаток, 20 джерел.

Об’єкт дослідження – комп’ютерні методи створення реалістичного оточення за допомогою математичних алгоритмів

Мета роботи – створити додаток з графічним дизайном та можливістю взаємодії з ним, генерація ландшафту на основі математичного методу реалізувати генерацію реалістичного ландшафту на основі вихідних даних з використанням JavaScript фреймворків.

Методи дослідження – математичні методи генерації текстур, ландшафтів, різноманітних ефектів процедурної генерації, а саме алгоритм шуму Перліна.

Результати – створений додаток з графічним дизайном та можливістю взаємодії з ним, генерація ландшафту на основі математичного методу реалізувати генерацію реалістичного ландшафту на основі вихідних даних з використанням JavaScript фреймворків.

ЗАСТОСУНОК, КОМП’ЮТЕРНА ГРАФІКА, ЛАНДШАФТ,
МАТЕМАТИЧНІ МЕТОДИ, ГЕНЕРАЦІЇ, JAVASCRIPT

Зміст

ВСТУП

За останні десятиліття індустрії кіно та відеоігор дуже сильно розвились і стали одними з найпередовіших індустрій в світі. Згідно з даними IDC, світовий дохід від відеоігор у 2020 році зріс на 20% до 179,7 мільярдів доларів, що зробило індустрію відеоігор більш прибутковою, ніж світова кіно та північноамериканська спортивна індустрія разом узяті. Згідно з даними Motion Picture Association, у 2019 році світова кіноіндустрія вперше досягла доходу в 100 мільярдів доларів, тоді як за оцінками PwC північноамериканський спорт принесе понад 75 мільярдів доларів у 2020 році. Нові технології дуже швидко починають використовуватись і перетворюють фільми та ігри у щось більше, ніж просто розвага, вони стають більш реалістичними та в найближчому майбутньому будуть дуже мало відрізнятись від реальності, це все буде доступне завдяки використання нових технологій та різноманітних алгоритмів. Технології, такі як VR, 3D без окулярів, доповнена реальність, візуалізація в реальному часі та інші [1]. VR – подібно до 3D-технологій, віртуальна реальність створена для підвищення залучення користувачів. Це приводить користувача прямо на край вашого сидіння, дозволяючи бачити картину з більшою сенсорною стимуляцією, ніж будь-коли раніше [2]. На даний момент VR-технологія все ще перебуває в зародковому стані, але в майбутньому вона обов’язково стане основною. Швидше за все, режисерам доведеться повністю переробити свою позицію щодо фільму, але для споживачів це буде того варте. Доповнена реальність є однією з нових технологій, які поєднують взаємодію в реальному часі між віртуальним і реальним світами, допомагаючи створювати точні тривимірні зображення [2]. По суті, ця технологія дозволяє аудиторії спостерігати накладені зображення реальних об’єктів. Кінематографісти на крок ближче до його впровадження, і завдяки технологічному прогресу він стане більш персоналізованим та інтерактивним [3].

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Процедурна генерація ландшафту, методи

Розробка ігор та фільмів стала дуже затратною через те, що розробники намагаються зробити картинку більш реалістичною. Все це потребує великих витрат на розробку. Для того, щоб зменшити ці витрати, часто почали використовувати алгоритми процедурної генерації.

Для початку треба визначити, що таке процедурна генерація, та що таке генерація цифрового вмісту. Генерація цифрового вмісту – це підмножина алгоритмів, вона є ключовою складовою сучасних методів створення фільмів, відеоігор, тобто того, що формується набором алгоритмів для створення ландшафту, карт, персонажів, зброї, різних ефектів: вогню, туману, дощу тощо, забезпечуючи зменшення витрат на виробництво [4].

Процедурна генерація – автоматичне створення контенту за допомогою алгоритмів. Іншими словами, процедурна генерація – це програмне забезпечення, яке самостійно може створювати унікальний контент. Під контентом у даному випадку розуміють генерацію цифрового вмісту, який був наведений вище. Ключовою особливістю створюваного контенту є те, що він повинен бути дійсним, тобто, якщо це гра, то згенерований контент не повинен заважати гравцеві пройти рівень, якщо фільм – то він повинен виглядати природно і т. д.

Як вже було сказано вище, у кінематографії, ігровій індустрії, а також освіті часто виникає необхідність моделювати чи генерувати навколишнє простір. Один із методів розв’язання даного завдання – це пагорбовий метод. Але цей метод не застосовується, коли потрібно деталізовано відобразити лише частину всього ландшафту. Також пагорбовим методом важко змоделювати схил пагорба, або гори, тому що в процесі генерації ландшафту використовуються лише гладкі півсфери. В основному для генерації

ландшафту використовується чотири основні алгоритму: шум Перліна, фрактальний та пагорбовий, ромбовидного квадрата[4].

Фрактальне стиснення – це метод стиснення цифрових зображень із втратами на основі фракталів. Цей метод найкраще підходить для текстур і природних зображень, покладаючись на той факт, що частини зображення часто нагадують інші частини того самого зображення. Фрактальні алгоритми перетворюють ці частини в математичні дані, які називаються «фрактальними кодами», які використовуються для відтворення закодованого зображення. Основна ідея фрактального методу полягає в рекурсивної генерації ділянки місцевості шляхом знаходження проміжних значень між двома заданими точками. Даний метод відрізняється математичною складністю, оскільки ділянки ландшафту доводиться розбивати на рівні частини, що не завжди можливо виконати наведено на рисунку 1.1 [5].

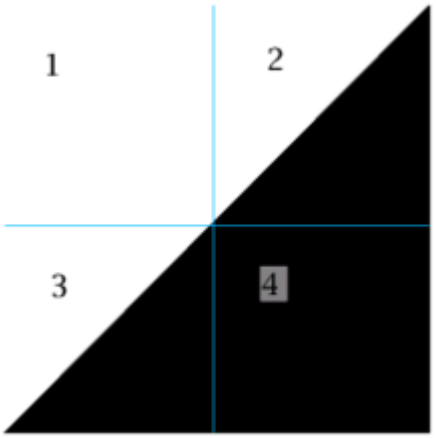

Рисунок 1.1 – Вигляд алгоритму фрактального стиснення

Пагорбовий метод відрізняється особливою простотою та легкістю реалізації. Його сутність полягає у генерації деякого кількості пагорбів у певній галузі. Змінюючи частоту виникнення і параметри, що визначають форму пагорба, можна налаштувати алгоритм для отримання ландшафту як рівнинної, і гірської місцевості. Також необхідно звернути увагу на те, що одержувані при цьому результати легко обчислюються та виглядають

реалістично. Розглянемо докладніше принципи роботи пагорбового методу При генерації ландшафт представляється суперпозицією пагорбових функцій, де кожен пагорб утворюється напівсферою. Для досягнення реалістичності вони іноді генеруються кількома проходами, висоти пагорбів першому проході з'єднуються, а карта висот кожного наступного проходу додається з картою висот попередніх. Звідси випливають проблеми методу: напівсфери гладкі, і тому отримання реалістичного ландшафту потрібна велика кількість проходів генерації, що значно ускладнює алгоритм та вимагає виконання великої кількості обчислень. Щоб позбутися надмірної гладкості, використовується рекурентний алгоритм, але при збільшенні ступеня його впливу на ландшафт виникає ймовірність появи аномалії. До того ж пагорбовий метод дуже важко адаптувати для генерації як самих гір, так і їх схилів.

Алгоритм «ромбоподібний квадрат» – це метод створення карт висот для комп'ютерної графіки. Алгоритм ромбовидного квадрата починається з двовимірного квадратного масиву шириною та висотою 2n + 1. Для чотирьох кутових точок масиву спочатку потрібно встановити початкові значення. Потім кроки ромба та квадрата виконуються по черзі, доки не буде встановлено всі значення масиву. Ромбовий крок: для кожного квадрата в масиві встановлюється середина цього квадрата як середнє значення чотирьох кутових точок плюс випадкове значення. Квадратний крок: для кожного ромба в масиві встановлюється середина цього ромба як середнє значення чотирьох кутових точок плюс випадкове значення. На кожній ітерації величина випадкового значення повинна бути помножена на 2-h, де h є значенням між 0,0 і 0,1 (нижчі значення створюють більш нерівний рельєф). Під час квадратних кроків точки, розташовані на краях масиву, матимуть лише три суміжні значення, а не чотири. Існує кілька способів впоратися з цим ускладненням – найпростішим є взяти середнє лише трьох суміжних значень. Іншим варіантом є «обгортання», беручи четверте значення з іншого боку

масиву. При використанні з узгодженими початковими значеннями кутів цей метод також дозволяє згенеровані фрактали зшивати без розривів.

На рисунку 1.2 показано кроки, пов’язані з виконанням алгоритму ромбоподібного квадрата на масиві 5 × 5 [6].

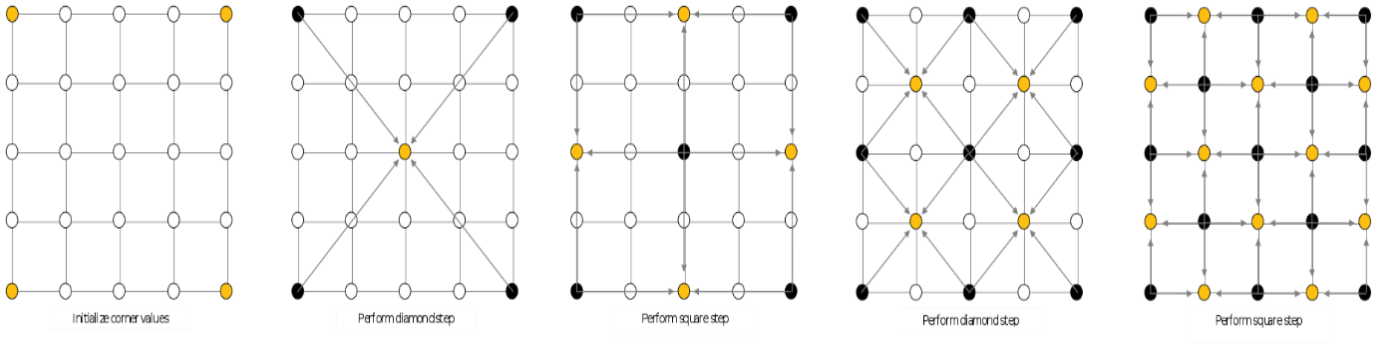

Рисунок. 1.2 – Вигляд алгоритму ромбовидного квадрата

На відміну від методів фрактального стискання та пагорбового, методи ромбоподібного квадрату та шуму Перліна є більш чіткими та менш вразливими до різних факторів. Через це для цієї роботи був обраний метод шуму Перліна.

1.2 Алгоритм шуму Перліна, шум, висота, частота, октави

Хороший генератор випадкових чисел видає числа, які не мають жодного зв’язку та не демонструють помітної моделі. Очевидно, трохи випадковості може бути корисною річчю при програмуванні органічної, реалістичної поведінки. Однак випадковість як єдиний керівний принцип не обов’язково є природним.

Існує алгоритм, який дає більш природні результати, відомий як «шум Перліна». Кен Перлін розробив функцію шуму під час роботи над оригінальним фільмом «Трон» на початку 1980-х років; він використовував її для створення процедурних текстур для комп’ютерних ефектів. У 1997 році Перлін отримав нагороду «Оскар» за технічні досягнення за цю роботу. Шум

Перліна можна використовувати для створення різноманітних ефектів із природними якостями, такими як хмари, пейзажі та текстури з візерунками, як мармур [7].

Шум Перліна має більш органічний вигляд, оскільки він виробляє природно впорядковану («гладку») послідовність псевдовипадкових чисел. На графіку нижче показано шум Перліна з плином часу, де вісь абсцис відображає час; зверніть увагу на плавність кривої наведено на рисунку 1.3.

Рисунок 1.3 – Послідовність псевдовипадкових чисел

Навпаки, на рисунку 1.4 показано чисті випадкові числа у часі.

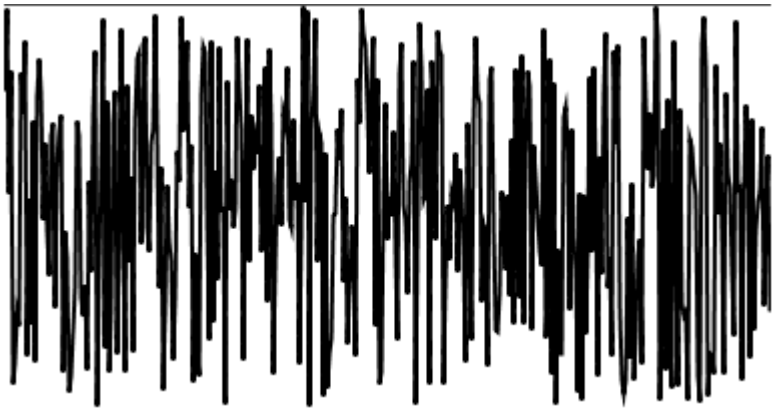

Рисунок 1.4 – Чисті випадкові числа в часі

Шум – це генератор випадкових чисел в комп’ютерній графіці. Це випадковий неструктурований шаблон, який добре підходить у цих випадках, коли потрібен джерело додаткових деталей, яких бракує в очевидній структурі.

У старих телевізорах, якщо налаштуватись на канал, на якому не було станції, можна побачити випадкові чорні та білі точки на екрані. Це шум. На радіо, якщо налаштуватись на канал, на якому немає станції, буде чутно шум (невідомого походження).

У обробці сигналу шум зазвичай є небажаним аспектом. У шумній кімнаті когось почути важче, ніж у тихій. Звуковий шум – це випадкові числа, розташовані в рядок (1D). На зображенні з шумами важче побачити шаблон, ніж на чистому зображенні. Шум зображення – це випадкові числа, розташовані в сітці (2D). Можна також мати шум у 3D, 4D тощо.

Багато природних систем виглядають шумно, тому, якщо при процедурній генерації чогось природного, зазвичай додають шум. Незважаючи на те, що реальні системи виглядають шумними, зазвичай існує структура, що лежить в основі; шум, який додають, не матиме такої самої структури, але це набагато простіше, ніж програмування симуляції, тому використовують його та сподіваються, що кінцевий користувач не помітить.

Найпростіший спосіб використовувати функцію шуму – використовувати її безпосередньо як висоту. Значення шуму безпосередньо використовується для висоти. Використання шуму зміщення середньої точки або шуму Simplex/Perlin як висоти також є прямим використанням. Інший спосіб використання шуму - це рух від попереднього значення. Наприклад, якщо функція шуму повертає [2, -1, 5], можна сказати, що перша позиція дорівнює 2, друга – 2 + -1 = 1, а третя – 1 + 5 = 6. Також можна зробити зворотне і використовувати різницю між значеннями шуму. Це також можна розглядати як модифікацію функції шуму [8].

Замість того, щоб використовувати шум для висоти, можна використовувати його для аудіо. Або, можливо, його використовують, щоб створити форму. Наприклад, можна використовувати шум як радіус на полярному графіку. Можна перетворити 1D-функцію шуму, у полярну форму, використовуючи результат як радіус, а не як висоту. Шум можна використовувати як графічну текстуру. Для цього часто використовується симплексний шум або шум Перліна. Можна використовувати шум, щоб вибрати розташування таких об’єктів, як дерева, золоті копальні, вулкани чи лінії розломів, що виникають під час землетрусів. Використовування шуму як порогову функцію. Наприклад, можна сказати, що кожного разу, коли значення перевищує 3, відбувається одне, інакше відбувається щось інше. Одним із прикладів цього є використання шуму 3D Simplex/Perlin для створення печер. Можна сказати, що все, що перевищує певний поріг щільності, є твердою землею, а все, що нижче цього порогу, є відкритим повітрям (печерою).

Шум Перліна – це багатомірний алгоритм, який використовується в процедурних генераціях, текстурах, генераціях рельєфа, генераціях карт, генераціях поверхонь, генераціях вершин і так далі, і тому подібне наведено на рисунку 1.5.

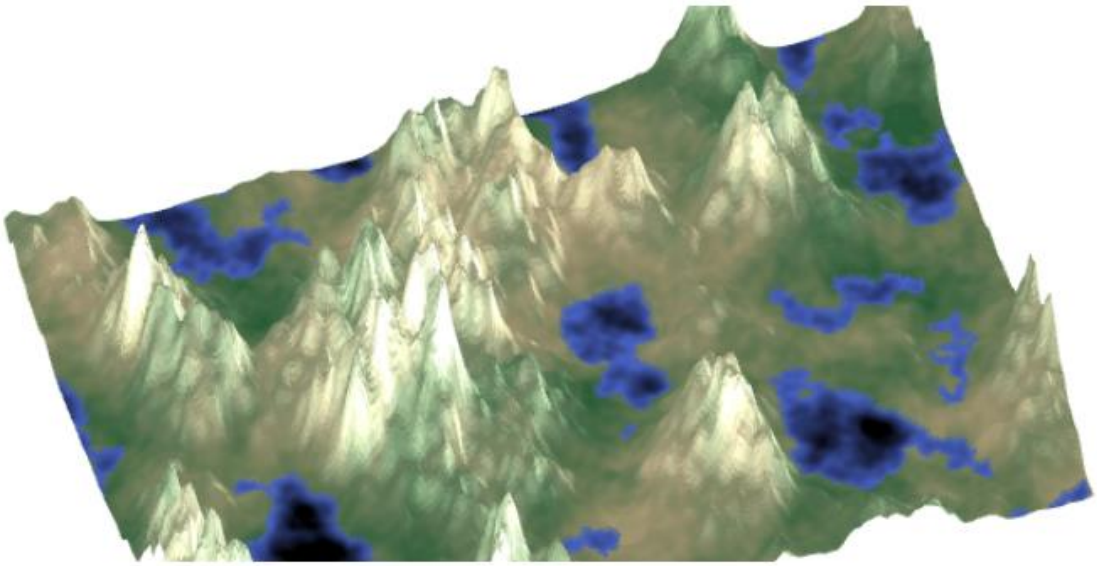

Рисунок 1.5 – Приклад роботи шуму Перліна

Стандартний спосіб генерації 2D-карт полягає у використанні як будівельний блок функції шуму з обмеженою смугою частот, наприклад шуму Перліна або симплексного шуму [9].

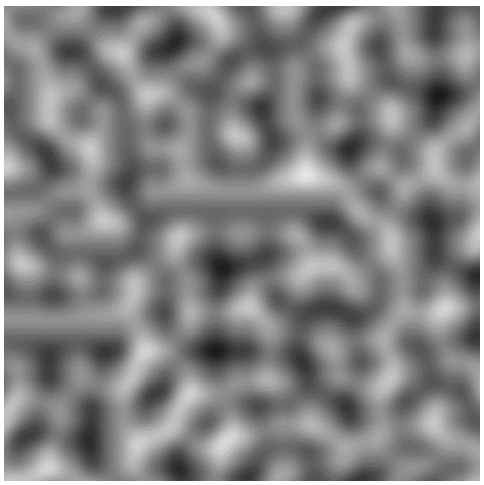

Рисунок 1.6 – Функція шуму

Присвоїмо кожній точці картки число від 0.0 до 1.0. У цьому зображенні 0.0 – це чорний колір, а 1.0 – білий.

Сам по собі шум – це просто набір чисел. І треба наділити його змістом. Перше, про що можна подумати це прив'язати значення шуму до висоти (це називається картою висот). Береться показаний вище шум і малюється він як висоти наведено на рисунку 1.7.

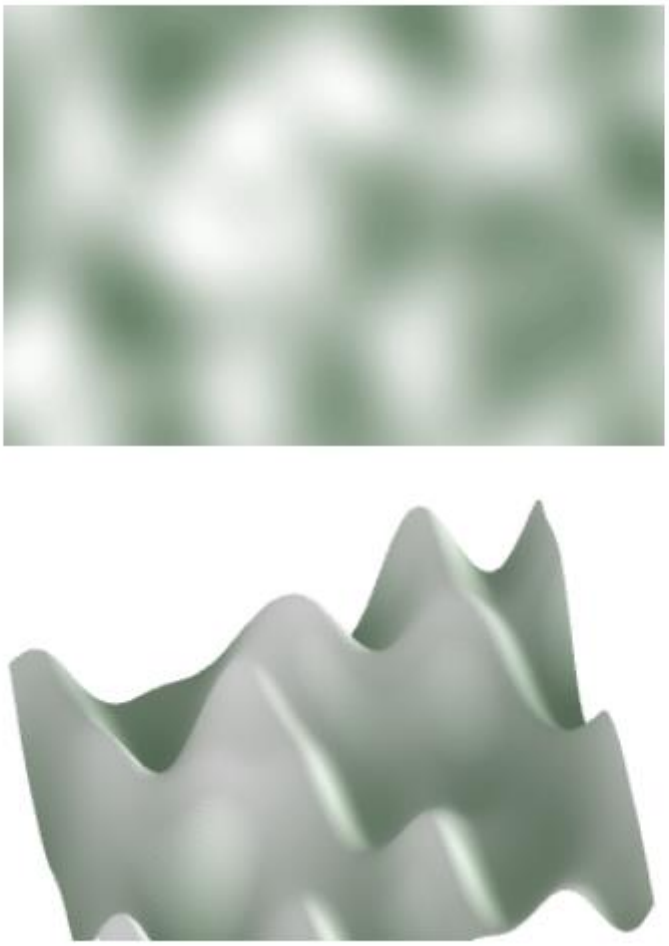

Рисунок 1.7 – Мапа висот

Частота – це основна властивість, на яку ми хочемо звернути увагу. Найпростіший спосіб зрозуміти це за допомогою синусоїд. Ось синусоїда нижчої частоти, за якою слідує синусоїда середньої частоти, за якою слідує синусоїда високої частоти наведено на рисунку 1.8.

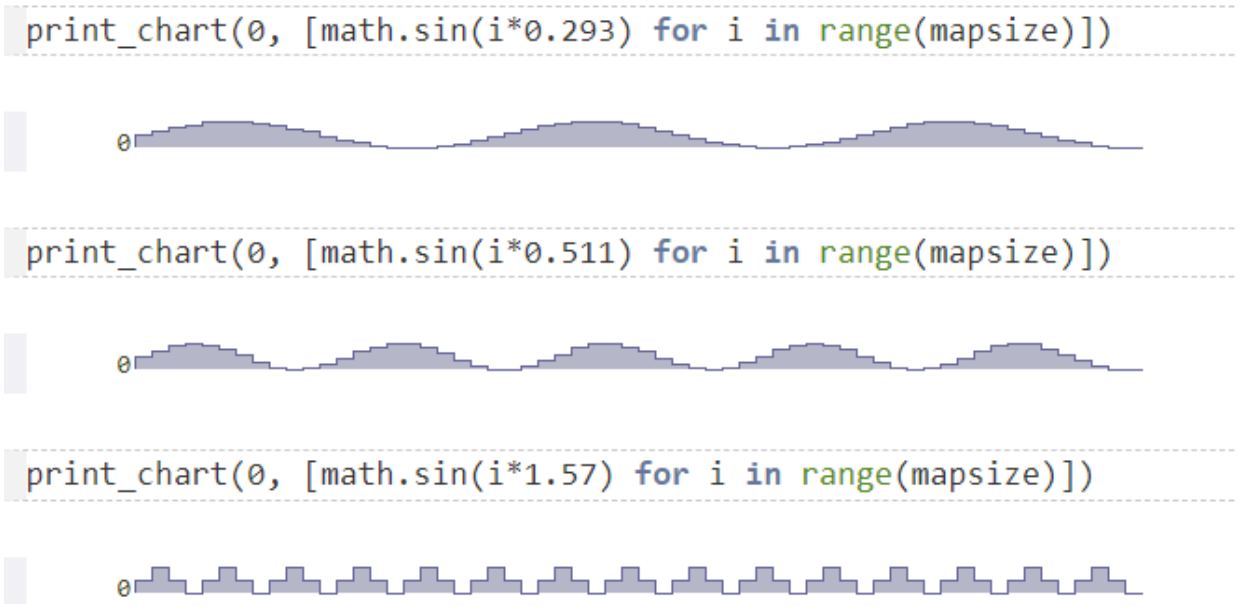

Рисунок 1.8 – Приклад частоти шуму

Ви бачите, що низькі частоти роблять пагорби ширшими, а вищі частоти роблять пагорби вужчими. Частота описує горизонтальний розмір ознак; амплітуда описує вертикальний розмір. Пам’ятаєте вище, коли я сказав, що карти долин/пагорбів/гір виглядають «занадто випадковими», і мені потрібні більші області долин або гір? По суті, я просив меншу частоту варіацій. Якщо у вас є безперервна функція, наприклад sin, яка створює шум, то збільшення частоти означає множення вхідних даних на певний коефіцієнт: sin(2*x) матиме подвоєну частоту sin(x). Збільшення амплітуди означає множення вихідного сигналу на певний коефіцієнт: 2*sin(x) матиме удвічі більшу амплітуду sin(x). У коді вище ви бачите, що я змінив частоту, помноживши вхідні дані на різні числа [10].

Також іноді корисно згадати про довжину хвилі, що є оберненою частотою величиною. При подвоєнні частоти розмір удвічі зменшується. Подвоєння довжини хвилі все вдвічі зростає. Довжина хвилі – це відстань, що

вимірюється в пікселях/тайлах/метрах або будь-яких інших одиницях, які ви вибрали для карт. Вона пов'язана із частотою: wave-length = map_size

Щоб зробити карту висот деталізованою, додається шум із різними частотами наведено на рисунках 1.9 та 1.10.

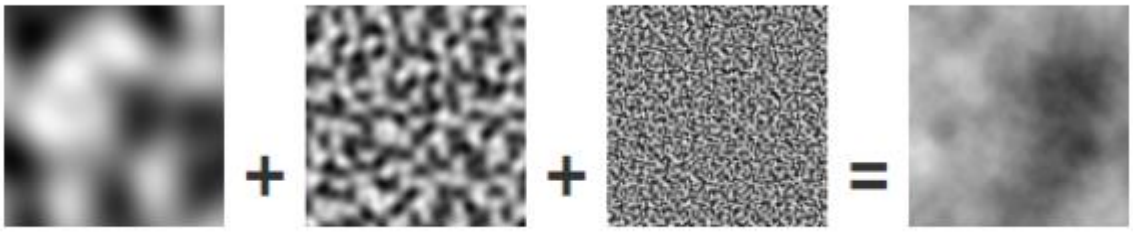

Рисунок 1.9 – Приклад частоти шуму

```
elevation[y][x] =    1 * noise(1 * nx, 1 * ny);
                  +  0.5 * noise(2 * nx, 2 * ny);
                  + 0.25 * noise(4 * nx, 2 * ny);
```

Рисунок 1.10 – Приклад зміни частоти шуму

Змішується в одній карті великі та низькочастотні пагорби з дрібними високочастотними пагорбами наведено на рисунку 1.11.

Рисунку 1.11 – Суміш на мапі великих низькочастотних пагорбів з дрібними високочастотними пагорбами

Функція шуму дає значення в інтервалі від 0 і 1 (або від -1 до +1, залежно від бібліотеки). Щоб створити рівнини плоскі, можна звести висоту в ступінь наведено на рисунку 1.12.

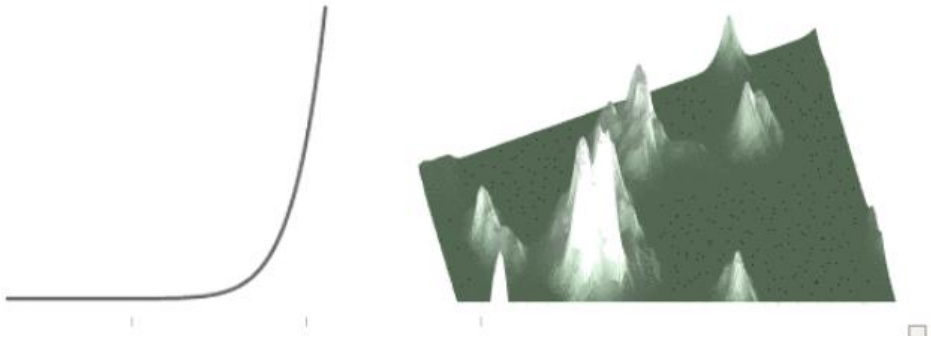

Рисунок 1.12 – Вигляд мапи після приведення до в ступінь

Високі значення опускають середні висоти рівнини, а низькі значення піднімають середні висоти убік гірських піків [11].

Шум дає цифри, але потрібна карта з лісами, пустелями та океанами. Перше, що можна зробити – перетворити малі висоти на воду наведено на рисунку 1.13

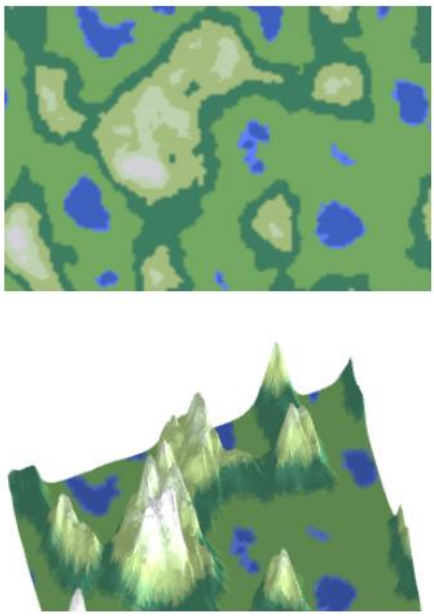

Рисунок 1.13 – Побудова рельєфу на основі мапи висот

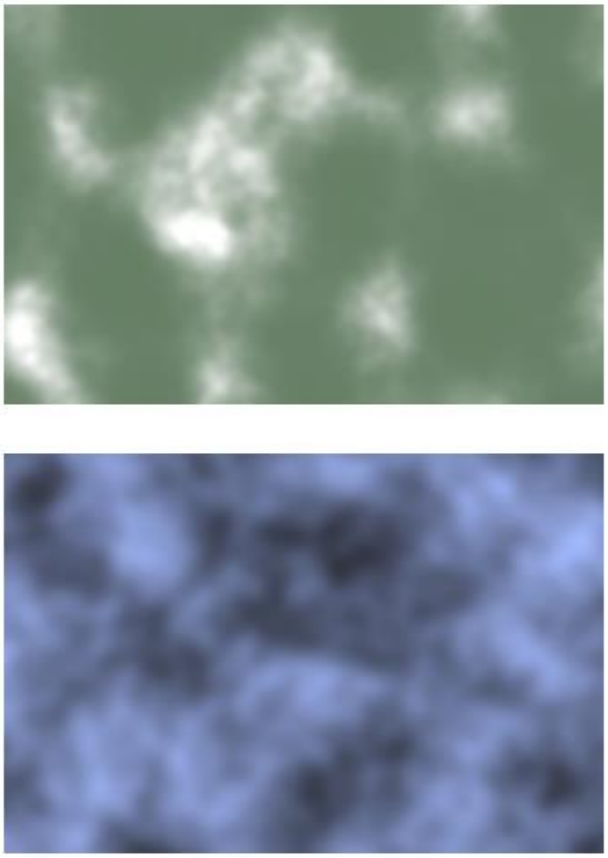

Рисунок 1.14 – Зверху – шум висот; знизу – шум вологості

Тепер використовується висота та вологість разом. На першому показаному нижче зображенні вісь y – це висота (взята із зображення вище), а вісь x – вологість (друге зображення вище рисунку 1.14).

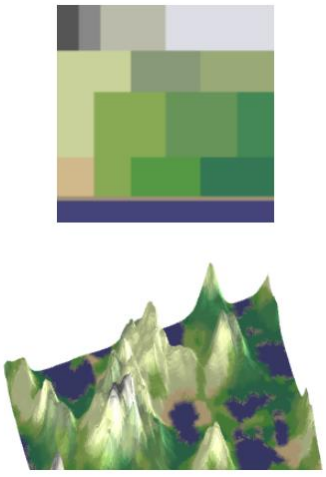

Рисунок 1.15 – Рельєф на основі двох значень шуму

Малі висоти – це океани та узбережжя. Великі висоти кам'янисті та засніжені. У проміжку між ними ми отримуємо широкий діапазон біомів наведено на рисунку 1.15.

Якщо округлити висоту до найближчих n рівнів, ми отримаємо тераси наведено на рисунку 1.16.

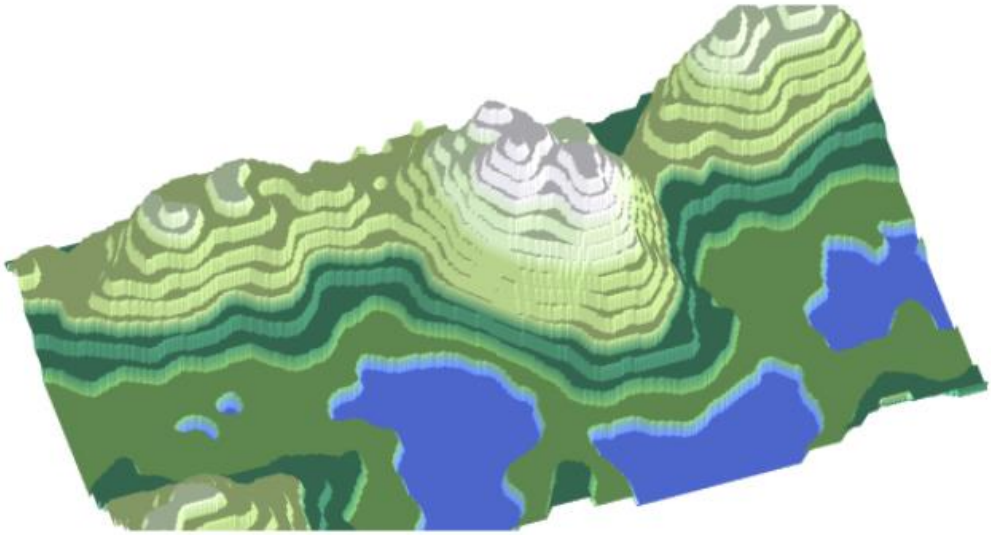

Рисунок 1.16 – Рельєф з терасами

Це результат застосування функції перерозподілу висот як e = f(e). Вище використовувалась e = Math.pow(e, exponent), щоб зробити різкішими гірські піки; тут використовується e = Math.round(e * n) / n для створення терас. Якщо використовувати не ступінчасту функцію, то тераси можуть бути округлішими або виникати тільки на деяких висотах [12].

1.3 Недоліки алгоритму Шуму Перліна

Шум Перліна, визначений у попередньому розділі, є алгоритмом для генерації плавної випадковості. Такі інструменти мають незліченну кількість застосувань у процедурній генерації, особливо в генерації рельєфу та синтезі текстур. Шум створює випадкові хвилеві шаблони в кількох вимірах, які ми можемо передати через різні математичні формули для досягнення

конкретних результатів. Perlin користується широкою популярністю в “шумовому просторі” протягом досить тривалого часу, аж до того, що він став частим у використанні у посібниках і бібліотеках. Шум Перліна помітно квадратичного зміщений. Він плавний і випадковий, як того вимагають формули, але особливості та форми, які створює шум, переважно вирівнюються під кутом 45 або 90 градусів до вхідних координат. Це явище зберігається глобальною протягом усього шуму, пропонуючи невелику прірву чи проміжок між ними. Така поведінка не підходить для програм цієї категорії, особливо якщо рішення має підтримуватися як стандарт. Алгоритм шуму Перліна заснований на квадратній, кубічній або загалом гіперкубічній сітці. На кожній вершині цієї сітки він вибирає напрямки нахилу (вектори градієнта), обчислює їх екстраполяції, а потім змішує їх разом із сусідніми. Його зміщення є прямим наслідком цієї структури сітки та того, як на ній взаємодіють градієнти. Нові градієнти з’являються лише у вершинах сітки, а їх виступи об’єднуються лише через вершини однієї комірки. Об’єкти з кращими кутами сітки об’єднуються без зусиль, тоді як ті, що мають інші кути, часто не настільки вдалі. Отримані візерунки прилягають до сітки, таким чином дотримуючись напрямків координат, які складають її основу [13].

1.4 Використання алгоритму Шуму Перліна

Майнкрафт – відеогра, розроблена Mojang Studios. Гру створив Маркус «Нотч» Перссон на мові програмування Java. Після кількох перших версій для приватного тестування її вперше оприлюднили в травні 2009 року, а потім повністю випустили в листопаді 2011 року, коли Нотч пішов у відставку, а Єнс «Джеб» Бергенстен взяв на себе розробку. З тих пір Minecraft був перенесений на кілька інших платформ і є бестселером відеоігор усіх часів, з понад 238 мільйонами проданих копій і майже 140 мільйонами активних гравців на місяць станом на 2021 рік [14].

За допомогою шуму Перліна створюються карти температури і вологості. Також на температуру впливає висота. Виключаючи болото, через кожен блок над рівнем моря (y = 64) температура стає нижче 1/600. Максимум впливає не тільки на карту біомов, але і на наявність осадків (За рахунок загрози, що виникає в будь-якій біомах Верхнього світу). На частоту випадіння осадків впливає вологість. В залежності від температури і вологості генерується карта біомов. Біом річки генерується на стиках двох біомов або з деяким шансом у межах одного біома наведено на рисунку1.17.

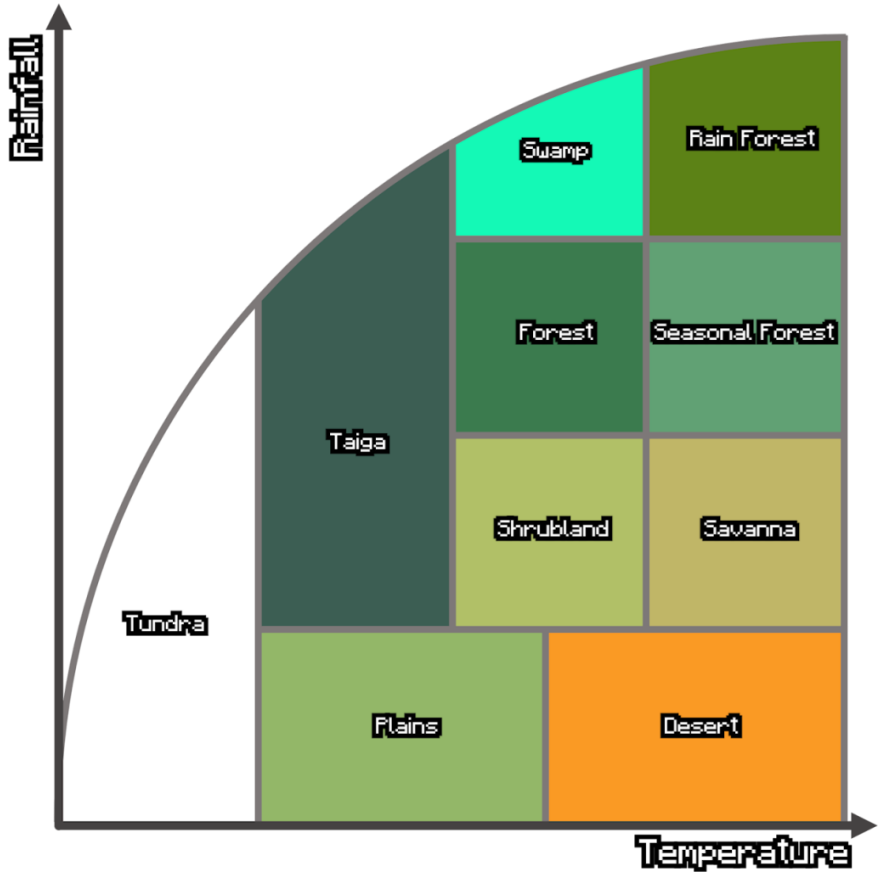

Рисунок 1.17 – Мапа температури і вологості

На сьогоднішній день гра перейшла на схожу систему, користуючись тривимірний шум Перліна. Тепер уже не генерується «висота землі». Значення шуму розглядається як «щільність», і всі блоки з щільністю менше 0 становляться повітрям, а блоки з щільністю більше або рівною 0 – землі. Щоб нижній шар був твердий, а верхній — ні, до отриманого результату додається висота (зміщення відносно рівня моря). Крім того, для поліпшення результату і прискорення обчислень використовується інтерполяція – світ ділиться на блоки 3×4×3, а після вирахування «щільності» для кожного з блоків результати зберігаються, генерація печери така ж, як і раніше додали нові блоки [15].

У 2013 році Genevaux et al. модифікували алгоритм та створили новий спосіб генерації ландшафту на основі гідрології. Їх алгоритм приймає на вхід

контури місцевості та річки на ньому, задані користувачем, алгоритм візьме вхідну інформацію починає генерувати повну дренажну мережу, що призводить до формування річок, які протікають від джерел до витоків, з очікуваним коливанням висоти. Після створення річок вони починають працювати з рельєфом, модифікуючи рельєф відповідно до річок, щоб загальний результат карти був правильним з топологічної точки зору [16].

1.5 Постановка роботи

Основуючись на проаналізованій інформації було вирішено створити інформаційну систему генерації рельєфу на основі математичного алгоритму шуму Перліна з графічним інтерфейсом за допомогою JavaScript фреймворків, що дозволить:

- створити користувацький інтерфейс;
- реалізувати процедурну генерацію рельєфу за допомогою шуму Перліна;
- додати функцію збереження заданих параметрів;
- додати функцію змінення вхідних даних ;
- додати функцію збереження функції шуму та введених користувачем даних;
- додати зміну масштабу та позиції камери;

Детальніше про методи вирішення і реалізацію буде написано у наступному розділі.

2 ІНФОРМАЦІЙНИЙ ОГЛЯД ПРОГРАМ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ

2.1 Обрання програмних методів реалізації

Існує багато методів для реалізації поставленої задачі, в деяких мовах програмування або середовищах розробки представлені зручні та швидкі методи для вирішення даної задачі, але серед усіх них була обрана мова програмування JavaScript.

JavaScript – це мова сценаріїв або програмування, яка дозволяє впроваджувати складні функції на веб-сторінках щоразу, коли веб-сторінка не просто стоїть там і відображає статичну інформацію, на яку користувач може дивитися, – відображаючи своєчасні оновлення вмісту, інтерактивні карти, анімовані 2D/ 3D-графіка, музичні автомати з прокручуванням відео тощо. Однак ще більш захоплюючим є функціонал, побудований на основі мови JavaScript на стороні клієнта. Так звані інтерфейси прикладного програмування (API) надають додаткові здібності для використання у власному коді JavaScript. API – це готові набори будівельних блоків коду, які дозволяють розробнику впроваджувати програми, які інакше було б важко або неможливо реалізувати. Вони роблять те саме для програмування, що й готові комплекти меблів для будівництва дому – набагато легше взяти готові панелі та з’єднати їх разом, щоб зробити книжкову полицю, ніж самостійно розробити дизайн, піти та знайти деревину, вирізати всі панелі потрібного розміру та форми, знайдіть шурупи потрібного розміру, а потім з’єднати їх, щоб зробити книжкову полицю. Загалом JavaScript перетворився на універсальну мову програмування завдяки різноманітним бібліотекам та фреймворкам. Зараз за допомогою цієї мови можна роботи не тільки клієнтську частину, а і серверну, розробляти десктопні програми та робити графічні додатки прямо у браузері. Для даної роботи були обрані три

найпопулярніші та дуже зручні фреймворки, а саме: React js, Node js та Tree js. Вони були обрані через те, що мають дуже зручний функціонал та повністю задовольняють потреби програми [17].

2.2 Опис обраних методів

React.js – це фреймворк і бібліотека JavaScript із відкритим кодом, розроблена Facebook. Він використовується для швидкого й ефективного створення інтерактивних інтерфейсів користувача та веб-додатків із значно меншою кількістю коду, ніж із ванільним JavaScript.

Розробка на React виконується за допомогою створення та повторно використовуючи компоненти, які можна розглядати як незалежні блоки Lego. Ці компоненти є окремими частинами кінцевого інтерфейсу, які, будучи зібраними, утворюють весь інтерфейс користувача програми.

Основна роль React у додатку полягає в тому, щоб обробляти рівень перегляду цього додатка так само, як view у шаблоні «model-view-controller» (MVC), забезпечуючи найкраще та найефективніше виконання візуалізації. Замість того, щоб мати справу з усім інтерфейсом користувача як єдиним блоком, React.js заохочує розробників розділити ці складні інтерфейси користувача на окремі багаторазово використовувані компоненти, які утворюють будівельні блоки цілого інтерфейсу користувача. При цьому фреймворк ReactJS поєднує швидкість і ефективність JavaScript з більш ефективним методом маніпулювання DOM для швидшого відтворення веб-сторінок і створення високо динамічних і адаптивних веб-додатків [18].

Як асинхронне середовище виконання JavaScript, кероване подіями, **Node.js** розроблено для створення масштабованих мережевих програм. Це на відміну від сьогоднішньої більш поширеної моделі паралелізму, в якій використовуються потоки ОС. Мережа на основі потоків відносно неефективна і дуже складна у використанні. Крім того, користувачі Node.js не хвилюються про блокування процесу, оскільки блокування немає. Майже

жодна функція в Node.js безпосередньо не виконує введення-виведення, тому процес ніколи не блокується, за винятком випадків, коли введення-виведення виконується за допомогою синхронних методів стандартної бібліотеки Node.js. Оскільки ніщо не блокує, масштабовані системи дуже зручно розробляти на Node.js [19].

Three.js – це платформа WebGL на основі JavaScript, за допомогою який можна запускати ігри та інші графічні програми прямо з браузера. Бібліотека three.js надає багато функцій і API для малювання 3D-сцен у вашому браузері. Three.js дозволяє створювати 3D-анімацію з прискоренням графічного процесора (GPU) за допомогою мови JavaScript як частину веб-сайту, не покладаючись на фірмові плагіни браузера. Це стало можливим завдяки появі WebGL, низькорівневого графічного API, створеного спеціально для Інтернету. Бібліотеки високого рівня, такі як Three.js або GLGE, SceneJS, PhiloGL та багато інших, дозволяють створювати складні тривимірні комп’ютерні анімації для відображення в браузері без зусиль, необхідних для традиційної автономної програми або плагіна [20].

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

В попередньому розділі були обрані методи для програмної реалізації, які зроблять розробку більш зручною та ефективною.

Програмна реалізація містить в собі:

– створення користувацького інтерфейсу;

– розроблення процедурної генерації рельєфу за допомогою шуму Перліна;

– додавання функції збереження заданих параметрів;

– додавання функції змінення вхідних даних;

– додавання функції збереження функції шуму та введених користувачем даних;

– зміну масштабу та позиції камери.

3.1 Опис алгоритму генерації та опис вигляду програми

Шум Перліна зазвичай реалізується як дво-, три- або чотиривимірна функція, але може бути багатовимірною. Зазвичай його реалізація включає три основні етапи: визначення сітки випадкових градієнтних векторів наведено на рисунку 3.18, обчислення скалярного добутку між градієнтними векторами та їх зсувами та інтерполяція між цими значеннями.

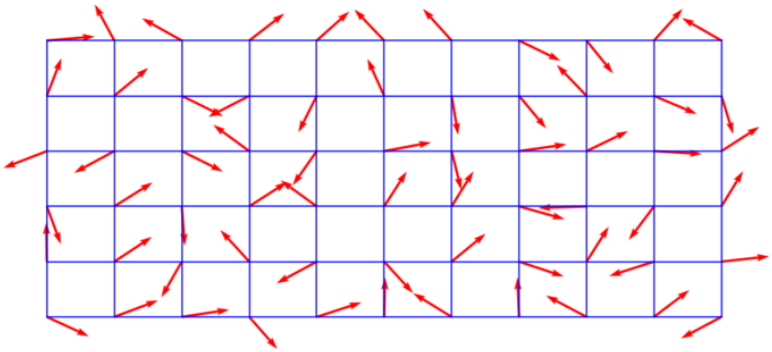

Рисунок 3.18 – Сітки випадкових градієнтних векторів

Визначається n-вимірна сітка, де кожен перетин сітки пов’язує з ним фіксований випадковий n-вимірний вектор градієнта одиничної довжини, за винятком одновимірного випадку, коли градієнти є випадковими скалярами від -1 до 1. Щоб розрахувати значення будь-якої точки-кандидата, спочатку знаходиться унікальна комірку сітки, у якій знаходиться точка. Потім визначається 2^n кути цієї клітинки та пов’язані з ними вектори градієнта. Далі для кожного кута обчислюється вектор зсуву. Вектор зсуву — це вектор зсуву від цього кута до точки-кандидата. Для кожного кута ми беремо скалярний добуток між його вектором градієнта та вектором зсуву до точки-кандидата. Цей скалярний добуток дорівнюватиме нулю, якщо точка-кандидат знаходиться точно в куті сітки. Зауважте, що вплив вектора градієнта зростає з відстанню, чого можна уникнути, спочатку нормалізувавши вектор зсуву до довжини 1. Це призведе до помітних різких змін, за винятком того, що відстань враховується на наступному кроці інтерполяції. Однак нормалізація вектора зсуву не є звичайною практикою. Для точки у двовимірній сітці це вимагатиме обчислення 4 векторів зміщення та скалярних добутків, тоді як у тривимірному вимірі це вимагатиме 8 векторів зміщення та 8 скалярних добутків. Загалом алгоритм має $O(2^n)$складність, де n— кількість вимірів.

Останнім кроком є інтерполяція між скалярними добутками 2^n. Інтерполяція виконується за допомогою функції, яка має нульову першу похідну (і, можливо, також другу похідну) у вузлах сітки 2^n. Тому в точках, близьких до вузлів сітки, вихідні дані будуть наближеними до скалярного добутку вектора градієнта вузла та вектора зсуву до вузла. Це означає, що функція шуму проходитиме через нуль у кожного вузла, що і наддасть шуму його типовий вигляд [21].

Нехай n = 1, функція, яка інтерполює значення a_0 у вузлі сітки 0 і значення a_1 у вузлі сітки 1 є

$$f(x) = a_0 + smoothstep(x) \times (a_1 - a_0) \text{ для } 0 \leq x \leq 1$$

де smoothstep – це функція, що використовувалася. Функції шуму в комп'ютерній графіці має значення в діапазоні [-1,0,1,0] і можуть бути масштабована відповідно результат інтерполяції наведений на рисунку 3.19.

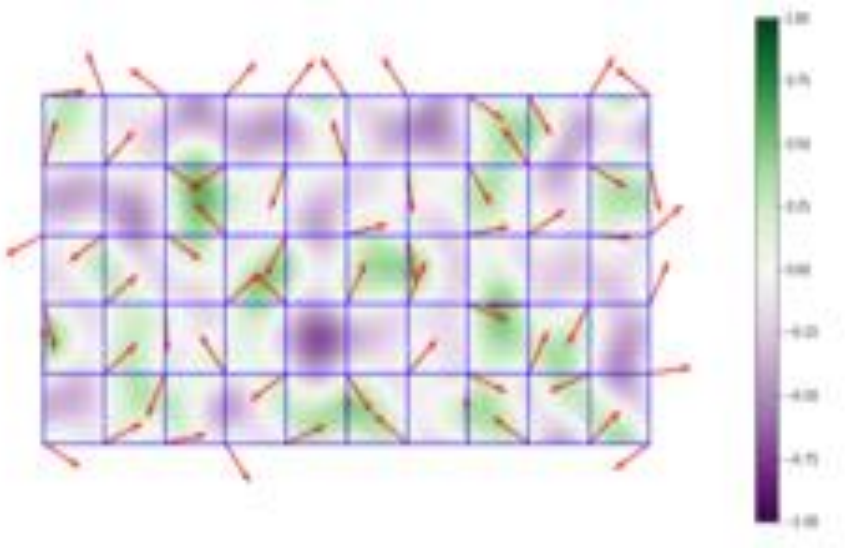

Рисунок 3.19 – Приклад результату інтерполяції

Перед початком розробки програми була розроблена структурна діаграма з основними діями на рисунку 3.20.

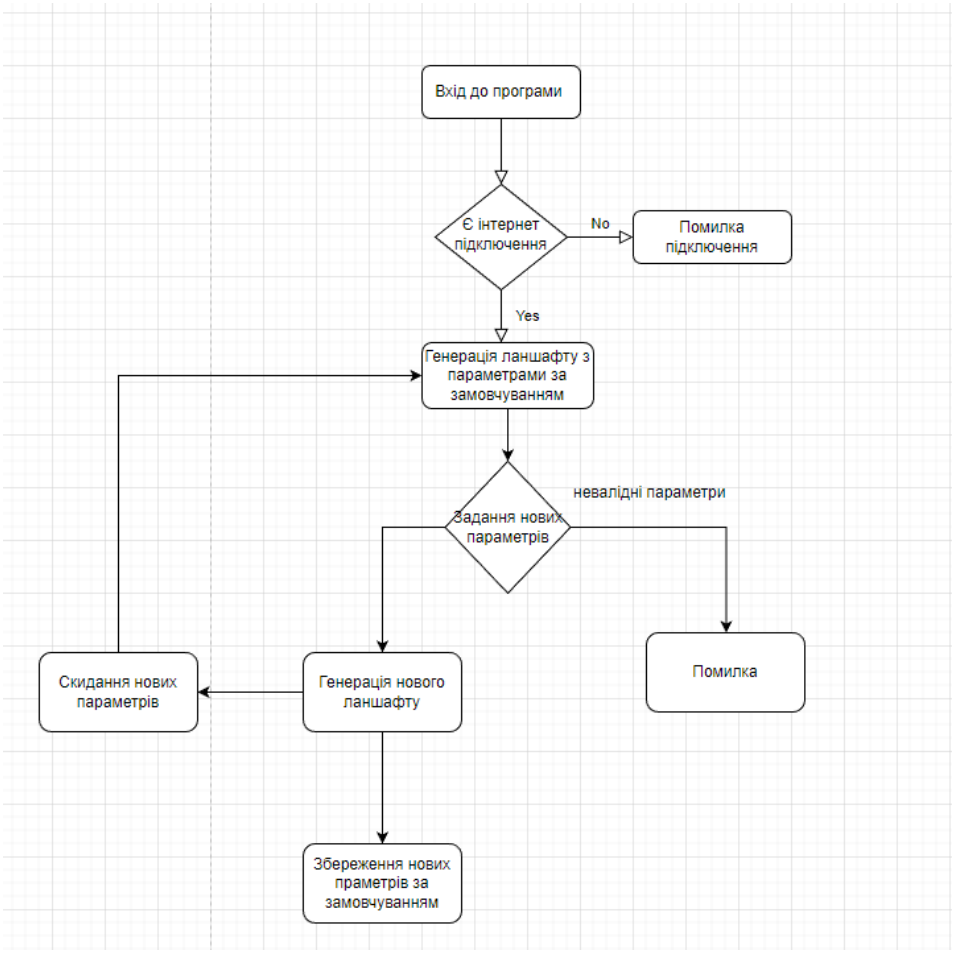

Рисунок 3.20 – Структурна діаграма проєкту

Також була розроблена use-case діаграма наведена на рисунку 3.21

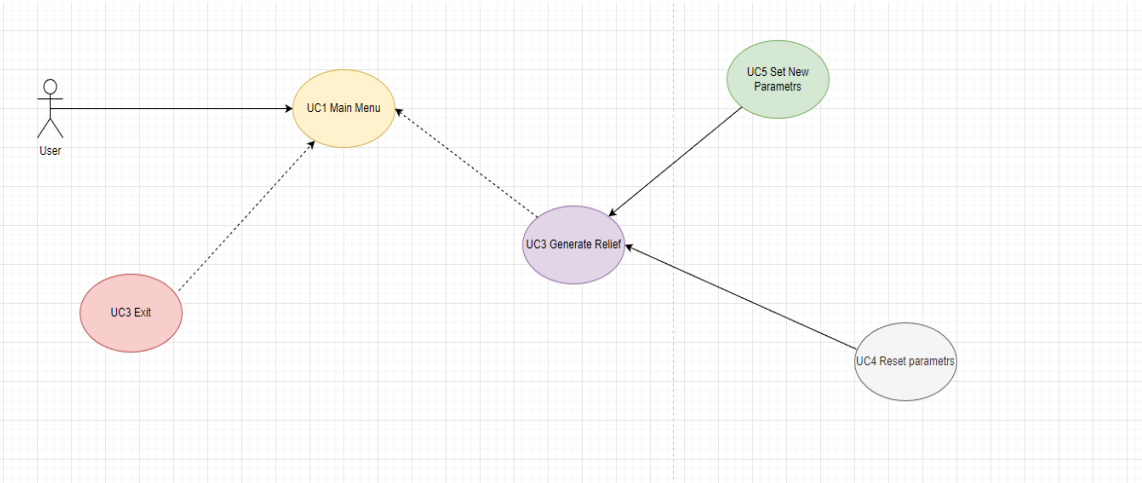

Рис. 3.21. Use-case діаграма

3.2 Інтерфейс та основні його можливості

Загальний вигляд меню налаштувань представлений на рисунку 3.22.

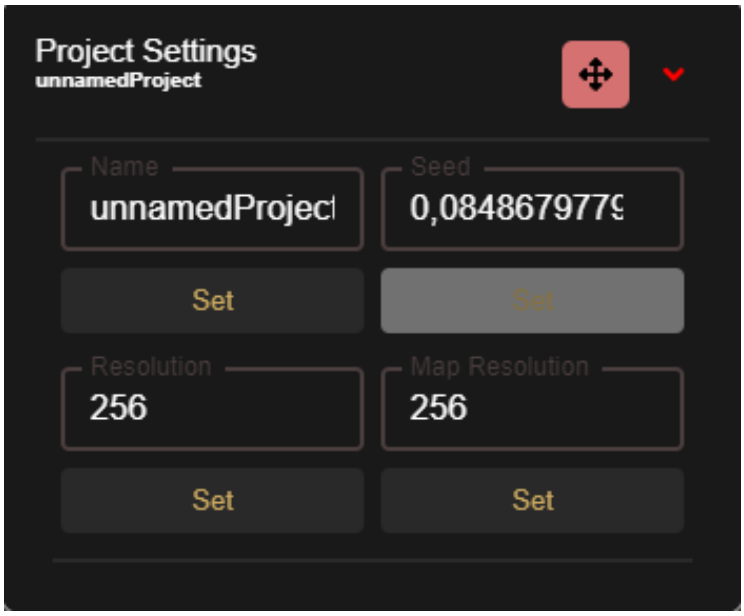

Рисунок 3.22 – Загальний вигляд меню налаштувань проєкту

Тут можна задати назву проекту, так зване зерно за рахунок якого генерується унікальний ландшафт, ще додатково зміна розміру мапи.

Окрім цього є ще окреме меню налаштування рельєфу є можливість зміни основних параметрів генерації ландшафту, також зображення мапи шуму можливість збереження мапи шумів та даних генерації, також задання нових параметрів генерації, чи скидання їх до тих що йдуть за замовчуванням приклад наведено на рисунку 3.23.

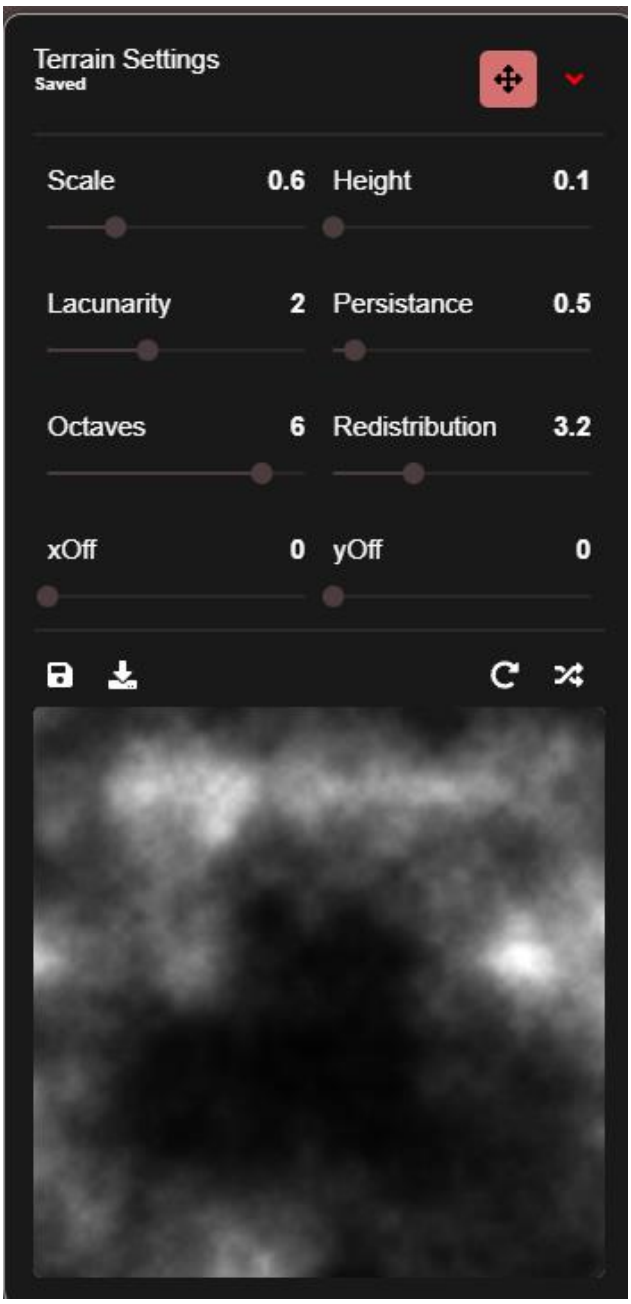

Рисунок 3.23 – Меню налаштування рельєфу

Також є можливість звернути меню або змістити його позицію. І головною частиною проекту є згенерований ландшафт на основі даних користувача, або даних за замовчуванням наведено на рисунку 3.24.

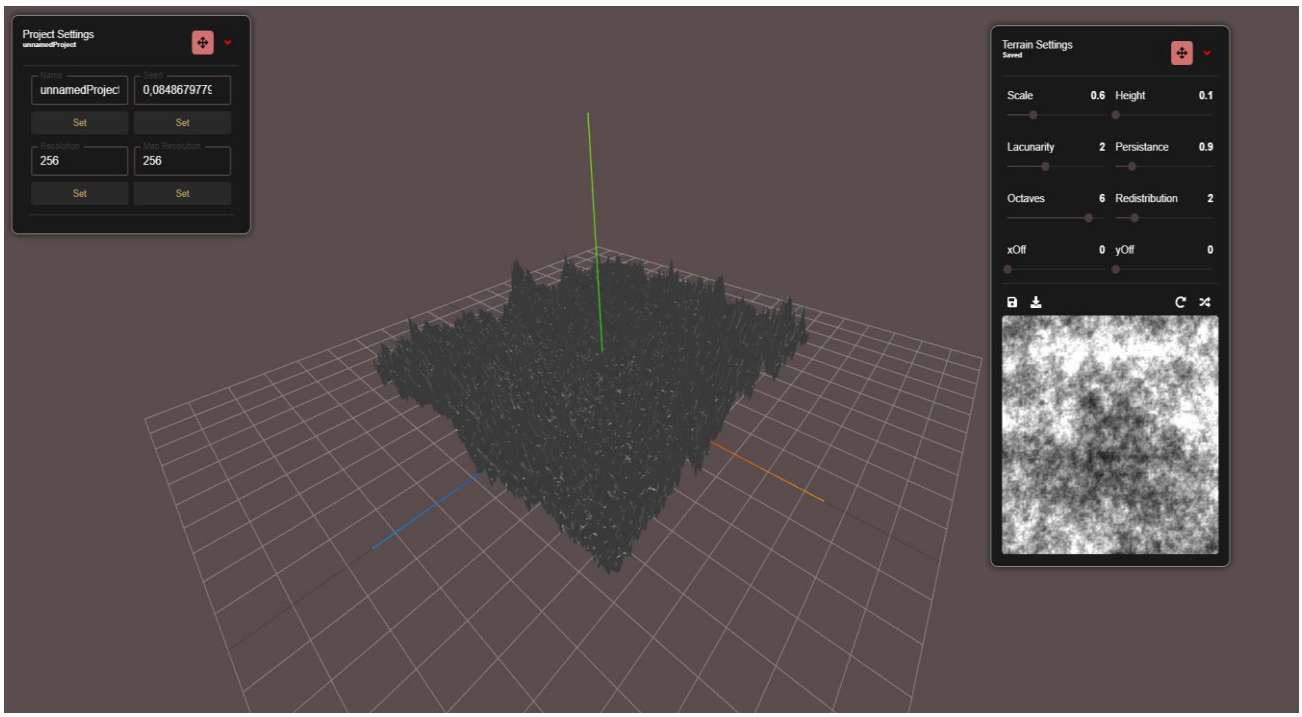

Рисунок. 3.24 – Вигляд згенерованої поверхні проєкту

ВИСНОВКИ

У роботі створена інформаційна система генерації рельєфу на основі математичного алгоритму шуму Перліна з графічним інтерфейсом за допомогою JavaScript фреймворків, а саме:

- створений користувацький інтерфейс;
- реалізована процедурна генерація рельєфу за допомогою шуму Перліна;
- додана функція збереження заданих параметрів;
- додана функцію змінення вхідних даних;
- додана функція збереження функції шуму та введених користувачем даних;
- додана зміна масштабу та позиції камери.

Усі поставлені завдання виконані автором. Основні результати роботи оприлюднені та обговорені на міжнародній науково-технічній конференції International Conference on Innovative Solutions in Software Engineering (ICISSE – 2022) (Івано-Франківськ, 2022 р.).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wallace Witkowski, “Videogames are a bigger industry than movies and North American sports combined, thanks to the pandemic - MarketWatch.” website. URL:https://www.marketwatch.com/story/videogames-are-a-bigger-industry-than-sports-and-movies-combined-thanks-to-the-pandemic-11608654990 (accessed Oct. 20, 2022).
2. Loring Weisenberger, “7 New Film Technologies Disrupting the Entertainment Industry | Wrapbook:” https://www.wrapbook.com/blog/new-film-technology (accessed Oct. 20, 2022).
3. Emily Moore, “6 Emerging Technologies Revolutionizing the Film Industry - Raindance.” website. URL: https://raindance.org/6-emerging-technologies-revolutionizing-the-film-industry/ (accessed Oct. 20, 2022).
4. Jessica Van Brummele “Procedural Generation.” website. URL: https://www.mit.edu/~jessicav/6.S198/Blog_Post/ProceduralGeneration.html (accessed Oct. 20, 2022).
5. “Алгоритм фрактального стиснення.” website. URL: https://www.wiki.uk-ua.nina.az/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D1%84%D1%80%D0%B0%D0%BA%D1%82%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE_%D1%81%D1%82%D0%B8%D1%81%D0%BD%D0%B5%D0%BD%D0%BD%D1%8F.html (accessed Nov. 10, 2022).
6. Steve Losh, “Terrain Generation with Diamond Square.” website. URL: https://stevelosh.com/blog/2016/06/diamond-square/ (accessed Nov. 9, 2022).
7. Adobe Inc, “Perlin Noise | Substance 3D Designer.” website. URL: https://substance3d.adobe.com/documentation/sddoc/perlin-noise-166363420.html (accessed Nov. 10, 2022).
8. Patricio Gonzalez Vivo & Jen Lowe, “The Book of Shaders: Noise.” website. URL: https://thebookofshaders.com/11/ (accessed Nov. 10, 2022).

9. Scratchapixel, “Perlin Noise: Part 2 (Perlin Noise).”// Scratchapixe URL: https://www.scratchapixel.com/lessons/procedural-generation-virtual-worlds/perlin-noise-part-2/perlin-noise (accessed Nov. 8, 2022).
10. Web Archive, “Generating Random Fractal Terrain.”// Web Archive. URL: https://web.archive.org/web/20170812230846/http://www.gameprogrammer.com/fractal.html (accessed Nov. 7, 2022).
11. ProcessingOrg,“Processing.org.”// Processing website. URL: https://processing.org/reference/noise_.html (accessed Nov. 6, 2022).
12. Jessica Van Brummelen and Bryan Chen, “Procedural Generation.” website. URL:https://www.mit.edu/~jessicav/6.S198/Blog_Post/ProceduralGeneration.html (accessed Nov. 5, 2022).
13. Amit Patel,“The Perlin Problem: Moving Past Square Noise” website. URL: https://noiseposti.ng/posts/2022-01-16-The-Perlin-Problem-Moving-Past-Square-Noise.html (accessed Nov. 10, 2022).
14. Minecraft Fandom, “History of world generation – Minecraft Wiki.”// MinecraftFandom”website.URL:https://minecraft.fandom.com/wiki/History_of_world_generation (accessed Nov. 10, 2022).
15. Clive Thompson, “The Minecraft Generation”// The New York Times Newspaper 2016 Page 49 of the Sunday Magazine with the headline: The Minecraft Generation, https://www.nytimes.com/2016/04/17/magazine/the-minecraft-generation.html (accessed Nov. 10, 2022).
16. J. D. Génevaux and B. Beneš, “Terrain generation using procedural models based on hydrology,” ACM Trans Graph, vol. 32, no. 4, Jul. 2013.
17. MDN Web Docs,“What is JavaScript? - Learn web development | MDN.”, URL:https://developer.mozilla.org/en/US/docs/Learn/JavaScript/First_steps/What_is_JavaScript (accessed Nov. 10, 2022).
18. David Herbert, “What is React.js? (Uses, Examples, & More).”, website. URL: https://blog.hubspot.com/website/react-js (accessed Nov. 10, 2022).

19. Node js Org,“About Node.js.”// Node.js Org” website. URL:https://nodejs.org/en/about/ (accessed Nov. 10, 2022).
20. Three.js Org, “Useful links – three.js docs.”// Three.js Org ” website. URL: https://threejs.org/docs/#manual/en/introduction/Useful-links (accessed Nov. 10, 2022).
21. Shivang Sarawagi, “Procedural Generation – A Comprehensive Guide Put In Simple Words - Scaleyourapp.”, website. URL: https://scaleyourapp.com/procedural-generation-a-comprehensive-guide-in-simple-words/ (accessed Nov. 10, 2022).

ДОДАТОК А
ЛІСТИНГ МОДУЛІВ ЕКСПЕРТНОЇ СИСТЕМИ

Файл App.js

```
import React, { Component } from 'react';

import "./App.css";

import ThreeDViewController from
'./Components/3dView/ThreeDViewController';



import TerrainSettingsRootViewController from
'./Components/ui/TerrainSettings/TerrainSettingsRootViewControll
er';

import ProjectInfoRootViewController from
'./Components/ui/ProjectSettings/ProjectInfoRootViewController';




export default class App extends Component {









  render() {

    return (

      <>
```

```
        <ThreeDViewController />
        <TerrainSettingsRootViewController />
        <ProjectInfoRootViewController />
      </>
    );
  }
}
```

Файл ThreeDViewController.js

```
import React, { Component } from 'react';
import ThreeDView from './ThreeDView';
import { main, refreshTerrain, rebuildTerrain } from
'./main/main';
import "./main.css";

import { connect } from 'react-redux';

class ThreeDViewController extends Component {
    componentDidMount() {
        main({...this.props})
    }
```

```
    componentDidUpdate(pProps) {
        if (Number(pProps.GEN_Resolution) !==
Number(this.props.GEN_Resolution)) {
            rebuildTerrain({...this.props})
        } else {
            refreshTerrain({...this.props})
        }
    }

    render() {
        return (
            <ThreeDView />
        );
    }
}

function mapStateToProps(state) {
    return {
        GEN_Resolution: state.GEN_Resolution,
        GEN_Seed: state.GEN_Seed,
        GEN_Scale: state.GEN_Scale,
        GEN_Persistance: state.GEN_Persistance,
        GEN_Lacunarity: state.GEN_Lacunarity,
```

```
        GEN_Octaves: state.GEN_Octaves,
        GEN_Redistribution: state.GEN_Redistribution,
        GEN_zScale: state.GEN_zScale,
        GEN_xOff: state.GEN_xOff,
        GEN_yOff: state.GEN_yOff
    }
}

export default connect(mapStateToProps)(ThreeDViewController)
```

Файл TerrainSettingsRootViewController.js

```
import React, { Component } from 'react';
import TerrainSettingsRootView from './TerrainSettingsRootView';

import "./main.css";

export default class TerrainSettingsRootViewController extends
Component {

    constructor(props) {
        super(props)
```

```
        this.state = {
            collapseOpen: true
        }
        this.onArrowPress = this.onArrowPress.bind(this)
    }

    onArrowPress = () => {
        this.setState({ collapseOpen: !this.state.collapseOpen
})
    }

    render() {
        return (
                <TerrainSettingsRootView
                    collapseOpen={this.state.collapseOpen}
                    onArrowPress={this.onArrowPress}
                />

        );
    }
}
```

Файл ProjectInfoRootViewController.js

```
import React, { Component } from 'react';

import ProjectInfoRootView from './ProjectInfoRootView';

import "./main.css";


import { connect } from 'react-redux';


class ProjectInfoRootViewController extends Component {


    constructor(props) {

        super(props)

        this.state = {

            collapseOpen: true,

        }

        this.onArrowPress = this.onArrowPress.bind(this)

    }


    onArrowPress = () => {

        this.setState({ collapseOpen: !this.state.collapseOpen
})

    }


    render() {
```

```
        return (
            <ProjectInfoRootView
                collapseOpen={this.state.collapseOpen}
                onArrowPress={this.onArrowPress}

                nameValue={this.props.META_ProjectName}
                resolutionValue={this.props.GEN_Resolution}
                seedValue={this.props.GEN_Seed}
                mapResValue={this.props.MAP_Resolution}
                mDate={this.props.META_mDate}

onResolutionChange={this.props.set_GEN_Resolution}

onProjectNameChnage={this.props.set_META_ProjectName}
                onMapResChange={this.props.set_MAP_Resolution}
                onSeedChange={(d) => this.props.set_GEN('Seed',
Number(d))}

                timeDisplace={this.props.TIME_displace}
            />
        );
    }
```

```
}



function mapStateToProps(state) {
    return {
        GEN_Resolution: state.GEN_Resolution,
        META_ProjectName: state.META_ProjectName,
        META_mDate: state.META_mDate,

        TIME_displace: state.TIME_displace,

        GEN_Seed: state.GEN_Seed,
        MAP_Resolution: state.MAP_Resolution
    }
}

function mapDispatchToProps(dispatch) {
    return {
        set_GEN_Resolution: (data) => dispatch({type:
'set_GEN_Resolution', data: data}),
        set_META_ProjectName: (data) => dispatch({type:
'set_META_ProjectName', data: data}),
```

```
        set_META_mDate: (data) => dispatch({type:
'set_META_mDate', data: data}),


        set_GEN: (label, data) => dispatch({ type: 'set_GEN',
data: data, label: label }),

        set_MAP_Resolution: (data) => dispatch({ type:
'set_MAP_Resolution', data: data})

    }

}


export default connect(mapStateToProps,
mapDispatchToProps)(ProjectInfoRootViewController)
```

Файл ProjectInfoRootViewController.js

```
import React, { Component } from 'react';

import ProjectInfoRootView from './ProjectInfoRootView';

import "./main.css";


import { connect } from 'react-redux';


class ProjectInfoRootViewController extends Component {


    constructor(props) {

        super(props)
```

```
        this.state = {
            collapseOpen: true,
        }
        this.onArrowPress = this.onArrowPress.bind(this)
    }

    onArrowPress = () => {
        this.setState({ collapseOpen: !this.state.collapseOpen
})
    }

    render() {
        return (
            <ProjectInfoRootView
                collapseOpen={this.state.collapseOpen}
                onArrowPress={this.onArrowPress}

                nameValue={this.props.META_ProjectName}
                resolutionValue={this.props.GEN_Resolution}
                seedValue={this.props.GEN_Seed}
                mapResValue={this.props.MAP_Resolution}
                mDate={this.props.META_mDate}
```

```
onResolutionChange={this.props.set_GEN_Resolution}


onProjectNameChnage={this.props.set_META_ProjectName}

                onMapResChange={this.props.set_MAP_Resolution}

                onSeedChange={(d) => this.props.set_GEN('Seed',
Number(d))}


                timeDisplace={this.props.TIME_displace}

            />

        );

    }

}




function mapStateToProps(state) {

    return {

        GEN_Resolution: state.GEN_Resolution,

        META_ProjectName: state.META_ProjectName,

        META_mDate: state.META_mDate,


        TIME_displace: state.TIME_displace,
```

```
        GEN_Seed: state.GEN_Seed,

        MAP_Resolution: state.MAP_Resolution

    }

}


function mapDispatchToProps(dispatch) {

    return {

        set_GEN_Resolution: (data) => dispatch({type:
'set_GEN_Resolution', data: data}),

        set_META_ProjectName: (data) => dispatch({type:
'set_META_ProjectName', data: data}),

        set_META_mDate: (data) => dispatch({type:
'set_META_mDate', data: data}),



        set_GEN: (label, data) => dispatch({ type: 'set_GEN',
data: data, label: label }),

        set_MAP_Resolution: (data) => dispatch({ type:
'set_MAP_Resolution', data: data})

    }

}


export default connect(mapStateToProps,
mapDispatchToProps)(ProjectInfoRootViewController)
```

Файл Erosion.js

```
let gl = undefined

let fbo = undefined


export function registerGLContext(context) {

    gl = context

}


export function registerFBO(_fbo) {

    fbo = _fbo

}


export function erode() {

    // let canvas = document.querySelector('.ui-map-canvas');

    // gl = canvas.getContext('webgl');


    const raw_heightBuffer = readPixels()

    console.log(raw_heightBuffer)

}


function readPixels() {

    gl.bindFramebuffer(gl.FRAMEBUFFER, fbo)

    var pixels = new Uint8Array(gl.drawingBufferWidth *
gl.drawingBufferHeight * 4);
```

```
    gl.readPixels(0, 0, gl.drawingBufferWidth,
gl.drawingBufferHeight, gl.RGBA, gl.UNSIGNED_BYTE, pixels);

    gl.bindFramebuffer(gl.FRAMEBUFFER, null)

    return pixels

}
```

Файл Island.js

```
export const Island = `


float makeIsland(vec2 p)

{

    float dist = distance(p.xy,vec2(0.0));

    float height = 1.0 - dist / 0.8;


    height = height < 0.0 ? 0.0 : height;


    return height;

}

` Файл Templates.js

import * as THREE from 'three';


export class BaseObject {

    constructor(geometry, material, options) {
```

```
        this.geometry = geometry

        this.material = material


        if(options.customMeshClass) {

            this.mesh = new
options.customMeshClass[0](...options.customMeshClass[1]);

        } else {

            this.mesh = new THREE.Mesh(this.geometry,
this.material);


        }


        this.mesh.name = options.name || 'Unnamed_Object'


        if(options.initRotation) {

            this.mesh.rotateX(options.initRotation.x)

            this.mesh.rotateY(options.initRotation.y)

            this.mesh.rotateZ(options.initRotation.z)

        }

        if(options.initTranslate) {

            this.mesh.translateX(options.initTranslate.x)

            this.mesh.translateY(options.initTranslate.y)

            this.mesh.translateZ(options.initTranslate.z)
```

```
        }

        this.modifiers = {}

    }

    getMesh() {
        return this.mesh
    }

    getName() {
        return this.mesh.name
    }

    getVerts() {
        return this.mesh.geometry.vertices
    }

    setVerts(verts) {
        if(typeof verts === Array) {
            if(typeof verts[0] !== THREE.Vector3) {
                throw new Error(`Verts must be of type
THREE.Vector3`)
```

```
            }
        }

        if(verts.length === this.mesh.geometry.vertices.length)
{
            this.mesh.geometry.vertices = verts
        } else {
            throw new Error(`Number of verticies does not match
orignal number.`)
        }

    }

    setName(name) {
        if(typeof name !== String) {
            throw new Error(`Name must be of type String.`)
        }
        this.mesh.name = name
    }

    addModifier(modifier, options) {
        let instance = new modifier(options)
        let name = String(instance.name)
```

```
        this.modifiers[name] = instance
    }

    removeModifier(mod) {
        for (const key in this.modifiers) {
            if (this.modifiers.hasOwnProperty(key)) {
                if (key === mod) {
                    this.modifiers[key] = undefined
                }
            }
        }
    }

}
```

Файл Axis.js

```
import { BaseObject } from '../templates/Templates';

const THREE = require('three')

export class AXIS extends BaseObject {
    constructor(options) {
        var size = options.size
```

```
        super(null, null, {
            name: options.name,
            customMeshClass: [
                THREE.AxesHelper,
                [size]
            ],
        })
    }
}
```

Файл Grid.js

```
import { BaseObject } from '../templates/Templates';


// import * as THREE from 'three';
const THREE = require('three')


export class GRID extends BaseObject {
    constructor(options) {
        var size = options.size
        var divisions = options.divisions


        super(null, null, {
```

```
            name: options.name,
            customMeshClass: [
                THREE.GridHelper,
                [size, divisions]
            ],
        })
    }
}
```

Файл main.js

```
import { BaseObject } from '../templates/Templates';

import * as THREE from 'three';
import { CustomShaderMaterial, TYPES } from  'three-custom-shader-material';

import { main, global } from '../Shaders/TerrainShaders/TerrainShders';

export class TERRAIN extends BaseObject {
    constructor(options) {
        const res_verts = options.resolution - 1
```

```
        const geometry = new THREE.PlaneBufferGeometry(1, 1,
res_verts, res_verts)


        var customUniforms = [
            THREE.ShaderLib.lambert.uniforms,
            {seed: {value: options.GEN_Seed}},
            {scale: {value: options.GEN_Scale}},
            {persistance: {value: options.GEN_Persistance}},
            {lacunarity: {value: options.GEN_Lacunarity}},
            {octaves: {value: options.GEN_Octaves}},
            {redistribution: {value:
options.GEN_Redistribution}},
            {zscale: {value: options.GEN_zScale}},
            {xoff: {value: options.GEN_xOff}},
            {yoff: {value: options.GEN_yOff}},


            {island:{value: false}},


            {diffuse: { value: new THREE.Vector3(1.0, 1.0, 1.0)
}},
            {emissive: {value: new THREE.Vector3(0.0, 0.0,
0.0)}},
            {opacity: {value: 1.0}}
        ];
```

```
        const material = new CustomShaderMaterial({
            baseMaterial: TYPES.PHYSICAL,
            vShader: [main, global],
            uniforms: customUniforms,
            options: {
                wireframe: false,
                side: THREE.DoubleSide,
                flatShading: true,
                lights: true,
                shadowSide: true,
            }
        }).getMaterial()



        super(geometry, material, {
            name: options.name,
            initRotation: new THREE.Vector3(-(Math.PI / 2), 0,
0),
        })

        this.material = material
    }
```

```
    updateUniforms(options) {
        for (const key in options) {
            if (options.hasOwnProperty(key)) {
                const element = options[key];
                const KEY = key.toLowerCase().split('_')[1]
                if(this.material.uniforms[`${KEY}`]) {
                    this.material.uniforms[`${KEY}`].value =
element
                }
            }
        }
    }
}
```

Файл Axis.js

```
import { RENDERER } from "./renderer";


// OBjects
import { GRID } from "./objects/Grid";
import { TERRAIN } from "./objects/Terrain";
import { AXIS } from "./objects/Axis";
```

```
let options = undefined

let renderer = undefined


export function main(args) {

    options = args

    renderer = new RENDERER()


    initGrid(renderer)

    initAxis(renderer)

    initTerrain(renderer)


}


function initGrid(renderer) {

    let gridObject = renderer.addObject(GRID, {

        name: 'Grid',

        size: 2,

        divisions: 20

    })


    return gridObject
```

```
}


function initAxis(renderer) {
    let axisObject = renderer.addObject(AXIS, {
        name: 'Axis',
        size: 0.75,
    })

    return axisObject
}


function initTerrain(renderer) {

    renderer.addObject(TERRAIN, {
        name: 'Terrain',
        resolution: options.GEN_Resolution,

        ...options
    })
}


export function refreshTerrain(args) {
```

```
    const terrain = renderer.objects.Terrain

    terrain.updateUniforms({...args})


}


export function rebuildTerrain(args) {

    renderer.removeObject('Terrain')

    renderer.addObject(TERRAIN, {

        name: 'Terrain',

        resolution: args.GEN_Resolution,


        ...args

    })


}
```

Файл render.js

```
import * as THREE from 'three';

import { OrbitControls } from
"three/examples/jsm/controls/OrbitControls";


export class RENDERER {

    constructor() {
```

```
        this.scene = undefined

        this.camera = undefined

        this.renderer = undefined

        this.planeMesh = undefined

        this.controls = undefined

        this.width = 1


        this.objects = {}


        this.initScene()

        this.costomizeRenderer()

        this.initLight()

        this.render()


        window.addEventListener('resize', () => {


            this.camera.aspect = window.innerWidth * this.width/
window.innerHeight;

            this.camera.updateProjectionMatrix();


            this.renderer.setSize(window.innerWidth *
this.width, window.innerHeight);
```

```
        }, false);
    }

    initScene() {
        this.scene = new THREE.Scene();
        this.camera = new THREE.PerspectiveCamera(
            75, window.innerWidth*this.width /
window.innerHeight, 0.1, 1000
        );

        this.renderer = new THREE.WebGLRenderer({ antialias:
true, alpha: true });
        this.renderer.setSize(window.innerWidth * this.width,
window.innerHeight);
        this.renderer.shadowMap.enabled = true

        this.renderer.domElement.style.outline = 'none'
        document.querySelector('.canavs-
root').appendChild(this.renderer.domElement)
        this.camera.position.set(0.75, 0.75, 0.75)

        this.controls = new OrbitControls(this.camera,
this.renderer.domElement);
```

```
        this.controls.enableDamping = true
        this.controls.dampingFactor = 0.25
        this.controls.enableZoom = false
        this.controls.enablePan = true
        this.controls.enableZoom = true

        this.controls.minPolarAngle = 0;
        this.controls.maxPolarAngle = Math.PI / 2;



    }

    initLight() {
        const light = new THREE.PointLight(0x404040, 2)
        light.position.set(-3, 10, -3)
        light.rotation.set(1, 1, 1)
        light.castShadow = true;
        //light.shadow.radius = 30;
        this.scene.add(light)

        const ambLight = new THREE.AmbientLight(0x404040, 0.9)
        this.scene.add(ambLight)
```

```
}

render = () => {
    window.requestAnimationFrame(this.render);

    this.controls.update()
    this.renderer.render(this.scene, this.camera);
}

costomizeRenderer() {
    let ele = this.renderer.domElement
    ele.className = "mainRenderer"
    ele.style.cursor = 'grab'
}

addObject(Obj, options) {
    let obj = new Obj(options)
    let name = options.name
    this.objects[name] = obj
    this.scene.add(obj.getMesh())
    return obj
```

```
    }


    removeObject(name) {
        const toDispose = this.scene.getObjectByProperty('name',
name);
        if (toDispose) {
            this.objects[toDispose.name] = undefined


            toDispose.geometry.dispose();
            toDispose.material.dispose();
            this.scene.remove(toDispose);
            return
        }
    }


}
```