Міністерство освіти і науки Україні

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ”

Кафедра мультимедії факультету інформатики

Кваліфікаційна робота

Освітній ступень – бакалавр

На тему “**Процедурна генерація віртуальних світів у ігровій індустрії**”

Виконав студент

4-го року навчання спеціальності

121 “Інженерія програмного забезпечення”

Сурженко В’ячеслав Олександрович

(ПІБ)

Керівник кваліфікаційної роботи:

Афонін А.О

(прізвище та ініціали)

Рецезент _____

(прізвище та ініціали)

Кваліфікаційна робота захищена

З оцінкою _____

Секретар ЕР _____

(підпис)

“_____” _____ 2023 р.

Київ – 2023

Графік підготовки кваліфікаційної роботи до захисту

Графік узгоджено “ ” 2022 р.

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової роботи	01.11.2022	
2.	Ознайомлення з темою кваліфікаційної роботи	03.12.2022	
3.	Розробка плану та	13.01.2023	

	структури роботи		
4.	Побудова технічного завдання	25.01.2023	
5.	Робота з науковою літературою	13.02.2023	
6.	Робота з практичною частиною	20.02.2023	
7.	Робота над текстовою частиною роботи	15.04.2023	
8.	Аналіз отриманих результатів з керівником	21.04.2023	
9.	Коригування роботи	13.05.2023	
11.	Захист роботи		

Студент Сурженко В.О

Керівник Афонін А.О

“____” _____

Зміст

Анотіція

У цій кваліфікаційній роботі розглянуті поняття процедурної генерації, віртуальних світів та методи генерування та використання процедурно згенерованих віртуальних світів у ігровій індустрії. Також порівняні методи генерації та на основі одного з них створений програмний застосунок для демонстрації роботи та подальшого використання.

Вступ

Ця кваліфікаційна робота є покращенням і розгорненням теми моєї минулорічної курсової роботи. За основу було взято поняття віртуальних світів, процедурної генерації та алгоритмів генерації та обробки світів для подальшого використання в ігровій індустрії.

Процедурна генерація досить актуальне у наш час питання, коли звичайних користувачів вже неможливо просто вразити невеликим, але власноруч створеним світом. Сучасні ігри та програми націлені на масштаб подій і для цього не вистачить тільки сили дизайнерів рівнів. В такий момент розробники звертаються до процедурної генерації 3Д світів і намагаються реалізувати алгоритми які надаються можливість кастомізування результату.

Для створення віртуальних 3Д світів не досить одного лиш алгоритму, потрібні вхідні данні, та детальний аналіз та обробка вихідного результату для того, щоб отримати потрібний дизайнерам світ. Сучасні системи надають можливості виконувати такі завдання, чим ринок зараз і користується.

Вже сьогодні, лідери серед розробки ігор пишуть та використовують існуючі алгоритми для того, щоб створити унікальний ігровий досвід для гравців та зменшити витрати на розробку. Існує безліч проектів які використовують процедурну генерацію, а сотні таких, які позиціонують генерацію, як основну ідею гри.

Мета цієї кваліфікаційної роботи – розглянути поняття віртуальних світів, процедурної генерації та популярні методи для генерації. Також

порівняти їх між собою, обравши один для подальшої реалізації як застосунка.

Текстова частина кваліфікаційної роботи складається з трьох розділів.

У першому розділі розбирається поняття віртуального світу та процедурної генерації з прикладами використання й проблемами, що вирішує.

У другому розділі розглянуто та описано найпопулярніші методи процедурної генерації:

- Алгоритм Diamond-Square
- Згортання хвильової функції
- Шум Перлина
- Клітинний автомат
- Алгоритм фрактального шуму

У третьому розділі порівняно всі описані методи процедурної генерації, обрано окремий і описана його реалізація за допомогою ігрового рушія Unity та мови C#. Створений застосунок генерує 4 різні типи мапи на основі вхідних значень користувача та демонструє принцип та метод роботи вибраного алгоритму генерації.

Мета

Метою кваліфікаційної роботи є дослідження та розробка методів процедурної генерації віртуальних світів для використання у відеоіграх. В рамках роботи розглянуто переваги та недоліки використання процедурної генерації в індустрії відеоігор та проаналізовано основні алгоритми процедурної генерації, їх роботу та реалізацію. Потім на основі отриманих знань буде розроблено програмний додаток на платформі Unity з використанням одного з алгоритмів процедурної генерації, зокрема шуму Перлина, за допомогою мови програмування C#. Результатом буде програмний продукт, який можна буде в подальшому вдосконалювати та використовувати для автоматичної генерації різноманітних віртуальних світів у відеоіграх.

Актуальність

Актуальність даної роботи можна поділити на два пункти:

- Процедурна генерація набирає популярності в ігровій індустрії завдяки таким перевагам, як ефективне використання ресурсів, унікальний ігровий процес і можливість швидко вести розробку. Однак використання процедурної генерації є складним і вимагає передових навичок розробки, що може стати перешкодою для деяких незалежних розробників і невеликих студій.

- Робота може бути корисна для розробників, які хочуть навчитися процедурній генерації та застосовувати її у своїх проектах. У роботі детально описано основні алгоритми процедурної генерації та їх реалізацію, а також надано готовий програмний додаток для використання в ігрових проектах. Робота також актуальна і корисна для вивчення та використання процедурної генерації в архітектурі, дизайні та інших сферах, де процедурна генерація може бути використана для створення унікальних об’єктів і форм. Хоча такі методи використання процедурної генерації в кваліфікаційній роботі не розглядаються.

РОЗДІЛ 1. Віртуальні світи

1.1 Що таке віртуальний світ

Віртуальний світ - це сфера інтернет-технологій, яка стала частиною нашого життя. Це віртуальне середовище, яке можено створювати за допомогою комп'ютера і доступу до Інтернету.

- *"Віртуальні світи - це постійні віртуальні середовища, в яких люди сприймають інших як присутніх поряд з ними і можуть взаємодіяти з ними*" – Ральф Шроедер, 2008 р [1] (переклад авторський)

- *"Віртуальні світи - це синхронна, постійна мережа людей, які представлені у вигляді аватарів і поєднані мережею комп'ютерів."* – Марк Белл, 2008 р [2] (переклад авторський)

- *"Віртуальні світи - це спільні простори, якими користується багато гравців одночасно. Вони містять графічний інтерфейс, який візуально зображує простір. Взаємодія у віртуальних світах відбувається в режимі реального часу, користувачі відчувають негайність. Віртуальні світи дають користувачам можливість змінювати світ, в якому вони перебувають; простір є інтерактивним та відбувається в децентралізованих світах за допомогою аватарів."* – Бітсі Бук, 2004 р [3] (переклад авторський)

Віртуальний світ - це світ, який існує в Інтернеті або на машині і може бути доступний для користувачів з усього світу або тільки локально. Він може бути створений як окрема програма, яка запускається на комп'ютері, або як частина онлайн-гри, соціальної мережі чи веб-сайту.

Віртуальний світ може мати вигляд ігрового простору, де користувачі можуть інтерактивно взаємодіяти з оточенням, об'єктами та іншими користувачами. Такі віртуальні ігрові світи можуть бути створені для розваги, навчання або бізнесу. Або це може бути просто ігрове середовище.

Також віртуальний світ може бути створений для спілкування з іншими людьми в інтернет-середовищі таких, як соціальні мережі або ж чат-кімнати, які можуть надати користувачам можливість взаємодіяти з іншими віртуально, обговорюючи теми та різну інформацію.

Віртуальний інтерактивний світ може також бути використаний для навчання та тренування. Як приклад, він може бути використаний для створення симуляцій, де студенти можуть відтворювати реальні ситуації та навчатись рішенням проблем без негативного впливу на реальність. Як приклад тренування пілотів чи космонавтів чи гонщиків.

Одним з найвідоміших прикладів віртуального світу є гра Second Life, яка створена для інтерактивного спілкування та розваг між юзерами. У ній юзер створює свій віртуальний аватар і майже живе інше життя.

Також віртуальний світ може стати засобом зняття стресу та відпочинку, адже він дозволяє користувачеві відволіктись від реальності й зануритись у віртуальну атмосферу, де йому не треба думати про проблеми реального життя. Це може бути особливо корисно для людей, які мають обмежені можливості у реальному світі, наприклад, людей з обмеженою мобільністю або тих, хто живе в ізольованому місці.

Варто пам'ятати, що віртуальний світ не повністю відображає реальність і може мати свої обмеження, недоліки. Наприклад, він не замінює реальних взаємодій та відносин з людьми, а також може викликати залежність та втрату контролю над власним часом та розсудом. Або може бути методом, для викачування грошей шахраями чи розробниками з користувачів.

У сучасному світі віртуальний стає все більш популярним та широко використовується у різних галузях та сферах, від розваг та комунікації до бізнесу й науки. Його потенціал необмежений, і з часом можна очікувати ще більш обширне та реалістичне використання цієї технології й також стрімкий розвиток самої технологій, особливо з розвитком віртуальної реальності.

1.2 Що таке процедурна генерація

Процедурна генерація - це технологія створення вмісту або об'єктів у комп'ютерних іграх та програмах шляхом автоматичного генерування їх з основних елементів або правил, що вводяться до процесу генерації.

Термін "процедурний" стосується конкретної процедури, тобто процесу, що виконується на комп'ютері для генерації. Основними характеристиками такої генерації є:

- Керованість. Керованість - це можливість контролювати алгоритм і підлаштовувати його під свої потреби. Залежно від завдання, необхідний час буде змінюватися, але, як правило, додаток повинен виконувати алгоритм і генерувати контент швидко.

-Надійність. Генератор повинен гарантувати достовірність кінцевого результату, відповідність між очікуванням та реальністю.

- Різноманітність. Створювати контент максимально різноманітний, не схожий один на одного, щоб досягти унікальних результатів.

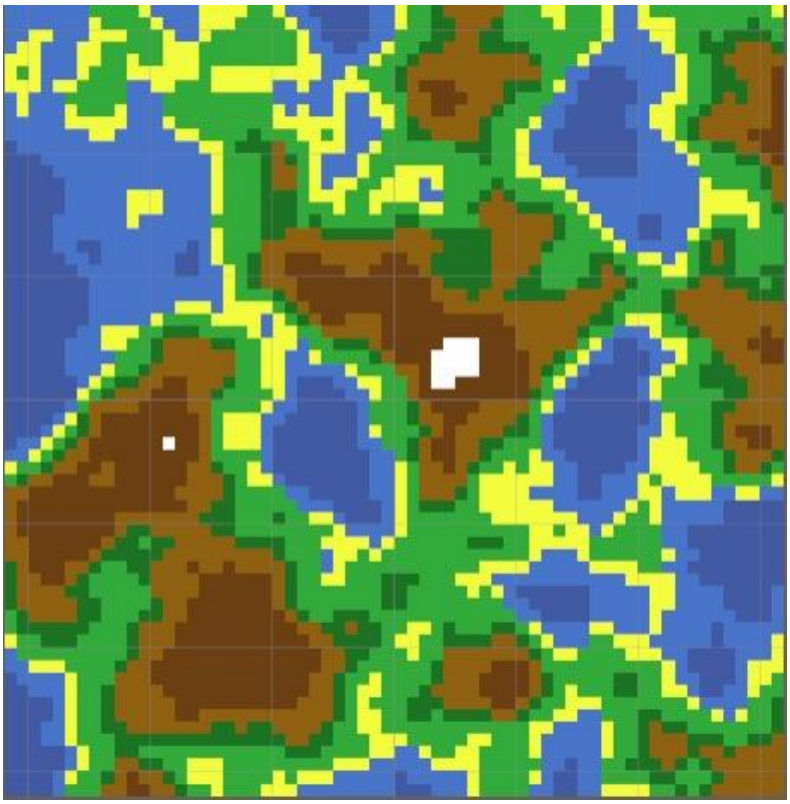

Рисунок 1.1 - Приклад процедурної генерації

- Правдивість. Створений контент повинен відповідати певним правилам і нормам, заздалегідь визначеним розробником.

У звичайних програмах та іграх, контент створюється розробниками вручну, що може зайняти багато часу розробки та зусиль команди. Процедурна генерація забезпечує створення великої кількості різноманітних об'єктів, таких як карти, міста, діалоги, зброя та інше, за допомогою алгоритмів та параметрів, заданих розробниками. Це дає можливість

збільшити кількість варіантів вмісту та зробити гру більш динамічною та цікавою.

Процедурна генерація використовується в різних жанрах ігор, наприклад, у відеоіграх-пісочницях, де генерація випадкових карт дозволяє створювати новий та унікальний досвід гравців кожного разу, коли вони грають. Вона також використовується в іграх з відкритим світом, де випадкові генерації карт дозволяють створювати нові локації та місця для дослідження, покращення існуючого ігрового процесу.

Процедурна генерація використовується у програмах комп'ютерної графіки, де вона дозволяє створювати складні об'єкти та ефекти, які було б складно або неможливо створити власноруч. Наприклад, візуалізація складних теренів у 3D-графіці може бути створена за допомогою процедурної генерації, або створення різних будинків та споруд.

Процедурна генерація є потужним інструментом, що надає можливість створювати велику кількість різноманітних об'єктів та вмісту у іграх і програмах, що робить їх цікавішими та динамічними для кінцевого користувача. Ця технологія стає здебільш популярнішою в індустрії відеоігор та комп'ютерної графіки, оскільки вона дозволяє розробникам ефективно та економно створювати складний та реалістичний вміст.

Варто зазначити, що процедурна генерація має свої обмеження та недоліки. Наприклад, згенерований контент може бути менш якісним та деталізованим, як власноруч створений контент. Також, використання процедурної генерації може призвести до створення вмісту, який не завжди

буде логічним або цікавим для гравців, або призводить до створення певних артефактів контенту, які можуть сильно вибиватися в контексту.

Отже, процедурна генерація - це потужна технологія, яка може значно збільшити кількість та різноманітність контенту у комп'ютерних програмах та іграх. Це дозволяє розробникам створювати цікавіші та динамічніші ігри, що можуть привернути більше уваги нових гравців. Однак, варто здійсняти контроль та забезпечити логічність й якість генерованого вмісту, щоб отримати максимально підходящій результат.

1.3 Проблеми, що вирішує генерація

Процедурна генерація - це технологія, яка дозволяє автоматично створювати контент: рівнини, міста, ландшафти та інші об'єкти в іграх та програмах. Ця технологія була розроблена для вирішення декількох проблем, пов'язаних зі створенням великої кількості контенту вручну та за короткий час.

Одна з головних проблем, яку вирішує процедурна генерація - це складність та час, необхідний для створення великої кількості контенту. Наприклад, створення складних рівнів може зайняти багато часу та ресурсів, тоді як процедурна генерація дозволяє створювати рівні легко та автоматично, за допомогою алгоритмів генерації й випадкових елементів. Також процедурна генерація може надати одразу декілька варіантів результату.

Ще одна проблема, яку вирішує процедурна генерація - це повторюваність вмісту. Часто в іграх є багато об'єктів, які повторюються:

дерева, камні та інші. Це може зменшити рівень реалістичності гри та зацікавленість гравців. Процедурна генерація дозволяє автоматично створювати різноманітні об'єкти та вміст для них, що забезпечує більшу різноманітність.

Крім того, процедурна генерація може допомогти зменшити вимоги до обсягу пам'яті та обчислювальних ресурсів. Створення великої кількості вмісту дизайнерами може призвести до збільшення обсягу пам'яті та навантажені обчислювальних систем, необхідних для виконання застосунка. Процедурна генерація дозволяє ефективніше використовувати пам'ять та обчислювальні ресурси, оскільки вона створює вміст залежно від потреб гри в конкретний момент часу і простору. Це дозволяє працювати на більшій кількості обчислювальних систем та зменшує потребу в спеціалізованих технічних вимогах.

Процедурна генерація може також викликати проблеми. Одна з них - це відсутність контролю над процесом створення контенту. Генерація може створювати випадковий та непередбачуваний вміст, який не завжди задовольняє потреби або користувачів та вибивається з всієї картини. Для того, щоб зменшити ризик такої генерації, розробники повинні добре розуміти технологію реалізованої в застосунку процедурної генерації та управляти її параметрами, щоб забезпечити бажаний результат. Інколи процедурно згенерований контент також підлягає ручній обробці.

Іншою проблемою, пов'язаною з процедурною генерацією, є відсутність унікальності вмісту. Хоча процедурна генерація дозволяє створювати різноманітний та неповторюваний контент, він може бути все ж

таки складеним до певної межі. Це означає, що багато ігор, що використовують процедурну генерацію, можуть мати схожий вміст або однакові рівні. Що призводить до зменшення зацікавленості гравців та привабливості контенту.

Процедурна генерація може допомогти вирішити багато проблем, пов'язаних зі створенням великої кількості вмісту вручну, але вона також може викликати деякі проблеми, що повинні бути враховані під час її використання.

1.4 Приклади використання генерації

Основною метою використання процедурної генерації - це створення контенту без використання людських ресурсів (Моделювальників, дизайнерів тощо), розробка різних типів світів та ігор, покращення реіграбельності, адаптація ігор під гравця, покращення контенту за допомогою алгоритмічних рішень та формалізація гейм дизайну з научної точки зору.

Перше задокументоване використання процедурно згенерованих світів датується 1980-ми роками. У той час обмежені ресурси ПК не дозволяли створювати різноманітні та масштабні ігрові світи. Типовим прикладом генерації є гра "Rogue", яка згодом започаткувала жанр "Roguelike", де

процедурно згенеровані світи є ключовою особливістю ігрового процесу.

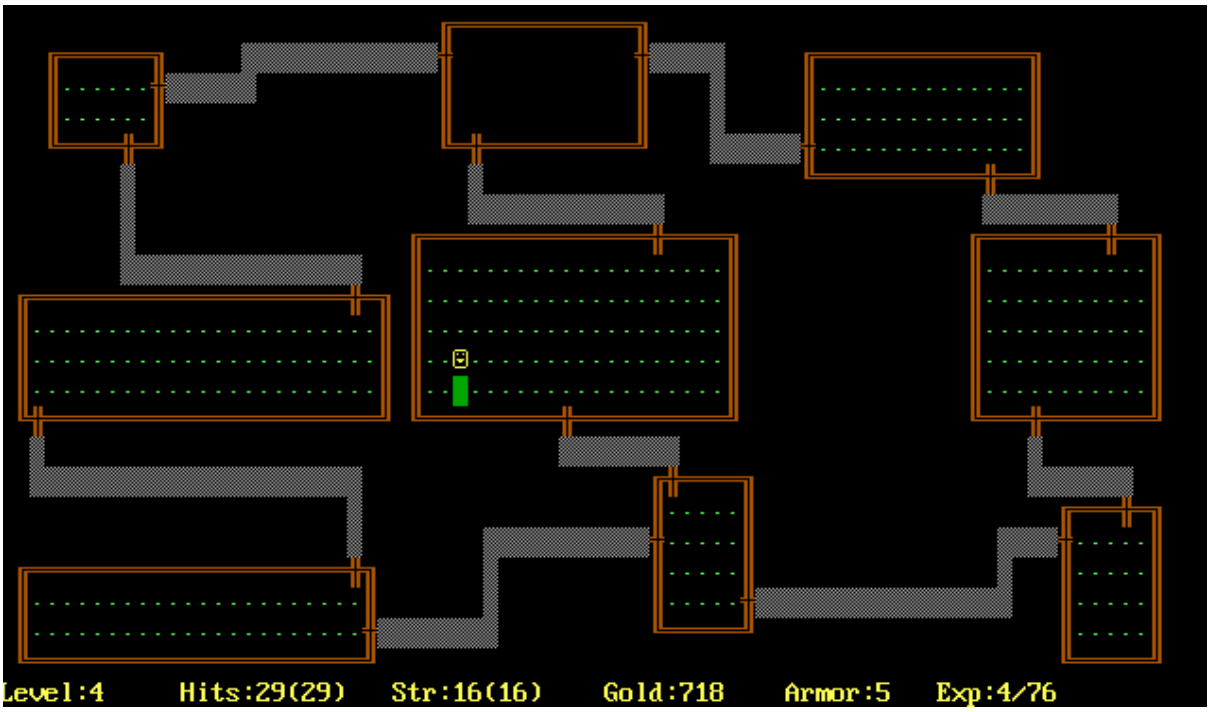

Рисунок 1.2 - Rogue

Нижче наведено список найпопулярніших ігор котрі використовують процедурну генерацію.

- Minecraft - це популярна відеогра, в якій світ створюється процедурно та в реальному часі. Кожен новий світ має унікальний ландшафт, рівні, поверхні й різноманітні блоки, що створюються за допомогою алгоритмів процедурної генерації.
- No Man's Sky - це космічна відеогра, в якій гравець може досліджувати нові зірки та галактики, зустрічаючи різноманітних інопланетян та відкривати нові планети. Генерація світу відбувається процедурно, створюючи унікальні планети та історії, які різняться для кожного

гравця. Розробники запевняють, що в грі генерується більше ніж 17 мільярдів планет та космічних тіл.

- Spelunky - це платформерна відеогра, в якій рівні створюються процедурно. Кожен новий рівень має унікальну форму, блоки та ворогів, що забезпечує велику кількість варіантів рівнів та зберігає гру захоплюючою для гравців. Кожнен новий старт в Spelunky суттєво відрізняться від минулого.
- Diablo III - це рольова відеогра, в якій рівні та зброя створюються процедурно. Це забезпечує велику кількість варіантів та зберігає гру захоплюючою для користувачів, оскільки вони не знають, що чекає їх в наступному рівні чи яку зброю або захист вони знайдуть.
- Dwarf Fortress - це стратегічна відеогра, в якій світ створюється процедурно, що дозволяє створювати унікальні географічні та кліматичні умови для гравців. Кожне місто, печера або країна та її історія у грі має своє власне унікальне середовище та історію, що робить гру захоплюючою та варіативною.

Загалом, процедурна генерація може бути використана для створення різноманітного контенту, що забезпечує велику кількість варіантів гри або програми, а також збільшує їх варіативність та зацікавленість для користувачів.

1.5 Висновок до розділу 1

У цьому розділі було розглянуто поняття віртуальних світів та процедурної генерації, та сфери їх використання. Було наведено та проаналізовано можливі покращення та проблеми які надає процедурна

генерація, було приведено пара прикладів використання процедурної генерації в ігровій індустрії.

РОЗДІЛ 2. Алгоритми генерації 3Д світів

2.1 Алгоритм Diamond-Square

2.1.1 Ідея та алгоритм

Алгоритм генерації ромбів (алгоритм Мандельброта) - це метод генерації реалістичних ландшафтів у комп'ютерній графіці та картографії. В основі алгоритму лежить ідея використання існуючих шаблонів для детального представлення згенерованої поверхні. Алгоритм складається з двох етапів: Алмазний та квадратний.

Алмазний етап:

На цьому етапі обчислюється середина кожної діагоналі квадрату. Наприклад, задано квадрат з координатами (x, y), (x + l, y), (x, y + l) і (x + l, y + l) і середина його верхньої діагоналі - ((x + l/2),y), а середина лівої діагоналі - координати (x, (y + l/2)). Висоти цих точок обчислюються як середнє арифметичне висот вершин, що вже існують на квадраті та його діагоналі.

Фаза квадрата:

На цьому етапі обчислюється центр кожної сторони квадрата. Наприклад, для квадрата з координатами (x,y), (x + l, y), (x, y + l), (x + l, y + l), середня точка верхньої сторони має координати ((x + l/2), (y + l/2)), а ліва середня точка має координати ((x +l/2), (y +l/2)). Висоти цих точок обчислюються як середнє арифметичне висот вершин, вже наявних у

квадратах, що прилягають до поточного квадрата. Цей процес повторюється для кожного згенерованого квадрата з меншими розмірами, поки не буде досягнута задана точність генерації світу.

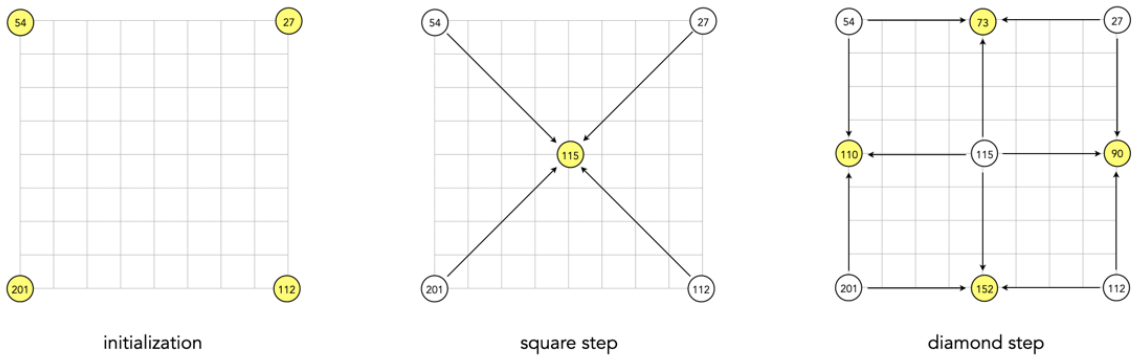

Рисунок 2.1 – Один крок алгоритму Diamond-Square

Реалізація алгоритму проходить таким чином:

1. Визначення початкових параметрів:
- Розмірність сторони квадрату
- Початкове значення висоти кутових точок
- Значення шуму
2. Генерація випадкових значень висоти в кутових точках квадрата, з використанням значення шуму.
3. Поки розмірність квадрату не досягне мінімальної значення, повторювати наступні кроки:
- Перевірити, чи розмірність поточного квадрата більша за мінімальну
- Виконати етап Diamond для кожного квадрату
- Виконати етап Square для кожного квадрату
- Зменшити розмірність квадрату вдвічі

- Зменшити значення шуму вдвічі

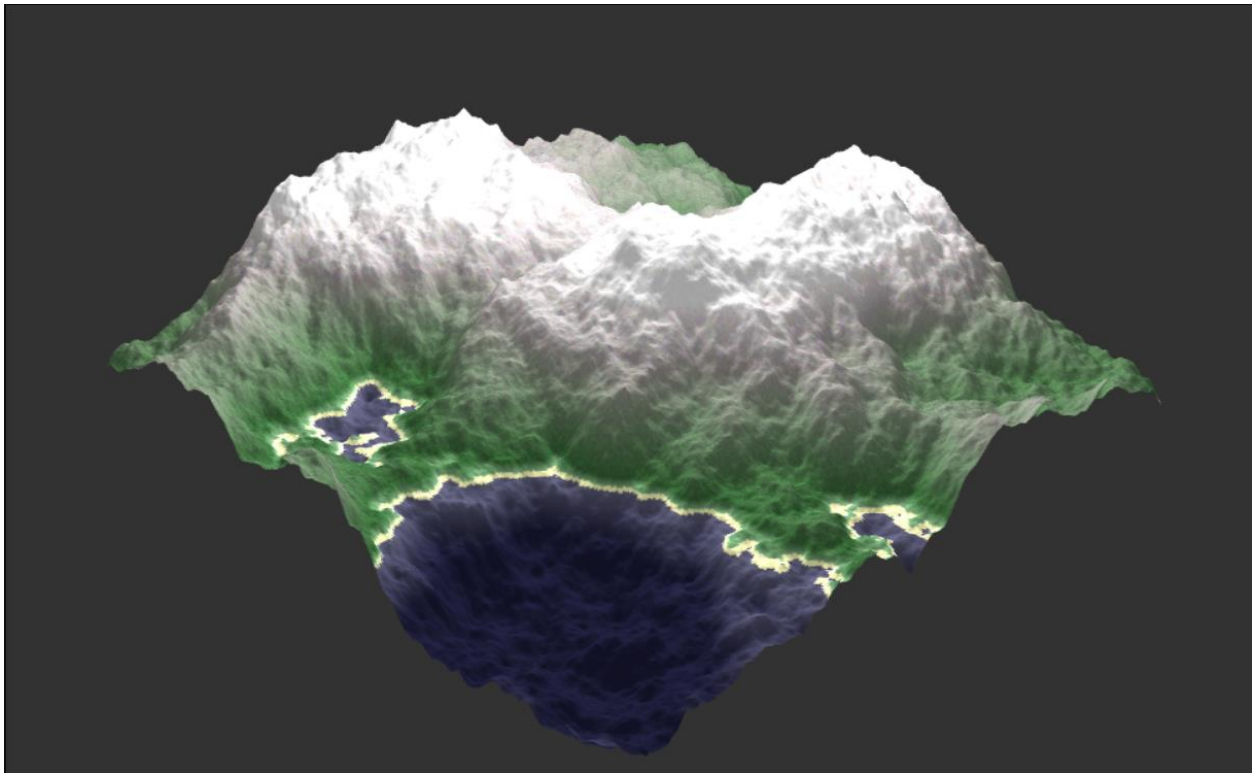

Рисунок 2.2 – Приклад згенерованого ландшафту

2.2 Згортання хвильової функції

Алгоритми згортки хвильових функцій (WFC) - це методи генерації зображень і дискретних структур, таких як музичні послідовності. Алгоритм використовується для створення реалістичних моделей на основі вхідних даних, які можна представити у вигляді сітки значень, наприклад, зображень або фрагментів тексту.

Основна ідея WFC полягає в тому, щоб знайти спільні характеристики між піделементами вхідних елементів і згенерованим результатом,

використовуючи імовірнісний підхід. Для цього алгоритм розбиває вхідну площину на блоки (наприклад, пікселі або плитки), кожен з яких має певні характеристики. Ці блоки використовуються для генерації нового зображення, де кожен блок у створеному зображенні має ті самі характеристики, що й відповідний блок у вхідному зображенні.

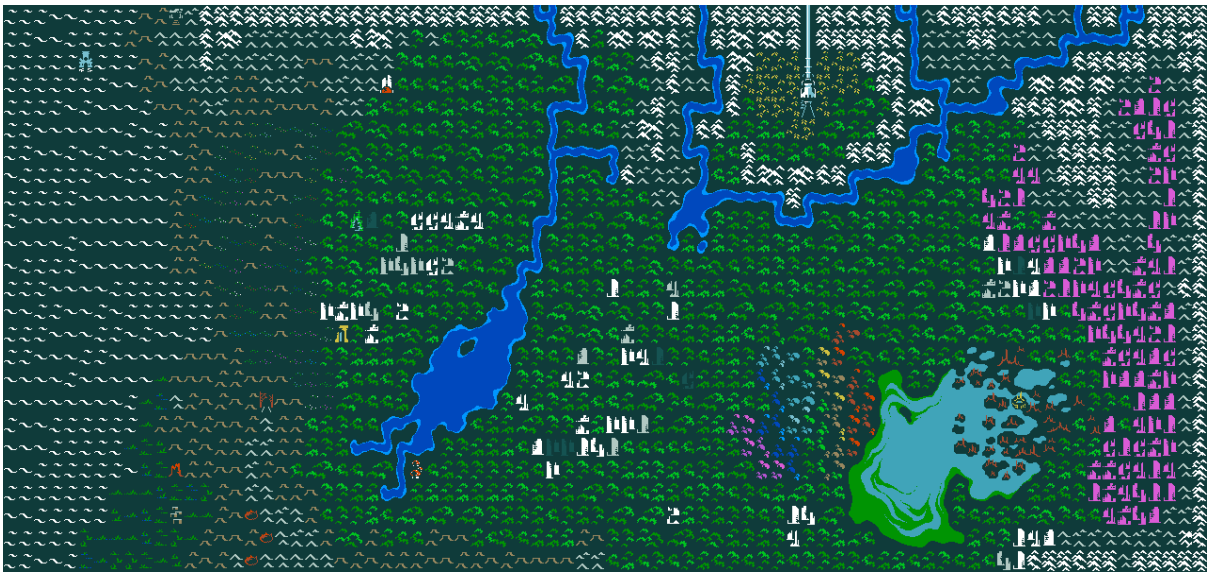

Рисунок 2.3 - Приклад роботи алгоритму згортання хвильової функції в грі Caves of Qud

Сам алгоритм може бути як суміжним, так і перекриваючим. Він також може виконуватися на 2D і 3D, гексагональних і нерегулярних сітках.

Алгоритм включає наступні кроки.

1. Підготовка даних: вхідні дані розбиваються на блоки з певними характеристиками, такими як колір, форма та орієнтація. Кожному блоку надається унікальний ідентифікатор.

2. Створення матриці суміжності: для кожного блоку створюється список можливих сусідніх блоків. На основі цього списку створюється матриця суміжності, в якій для кожного блоку відображаються можливі сусідні блоки та їхні ймовірності потрапляння.

3. Побудова початкового стану системи: створюється випадковий початковий стан системи і кожен блок на згенерованій поверхні може мати будь-яке значення з набору можливих. Іншими словами на площину наноситься випадковий елемент у випадковому місці.

4. Застосування алгоритму WFC: на кожному кроці алгоритму зі стану системи вибирається блок з найменшою ентропією (тобто найменшою кількістю можливих значень) і знаходиться його значення на основі ймовірності можливих значень, зазначених у матриці суміжності. Потім алгоритм поширює це значення на сусідні блоки, які ще не мають визначеного значення, і оновлює значення елементів новим знайденим значенням.

5. Повторюємо крок 4 до тих пір, поки всі блоки на згенерованому просторі не будуть ідентифіковані.

6. Отримання згенерованого світу: оскільки кінцевий стан системи містить повністю визначені значення для всіх блоків світу, можна побудувати згенерований світ шляхом об'єднання всіх блоків.

Алгоритм застосовується не тільки до генерації світів, але й до генерації дискретних структур, таких як плиткові картинки, мозаїки, музичні послідовності тощо. Він використовується в області генерації та дизайну

зображень, наприклад, в комп'ютерній графіці, де потрібно генерувати велику кількість реалістичних зображень.

Рисунок 2.4 - Робота Алгоритму ЗХФ у грі Bad Norse

2.3 Шум Перлина

Алгоритм генерації шуму Перлина - це метод створення нерегулярних текстур і зображень, що імітують природні процеси за допомогою випадкового шум. Шум Перлина був розроблений Кеном Перлином у 1983 році і досі широко використовується в комп'ютерній графіці, анімації, візуалізації даних та комп'ютерних іграх.

Основний принцип алгоритму полягає у накладанні шуму відповідної частоти на віртуальну поверхню розміром N x M. За замовчуванням, шум, що

створюється алгоритмом Перлина, має три октави, тобто три рівні деталізації зображення, додані один до одного. Перша октава містить найгрубіші деталі зображення, друга - середні, а третя - найдрібніші.

Алгоритм виробляє градієнт чисел, який можна використовувати для створення карти висот. Карта буде виглядати як карта у градації сірого, де світліші ділянки представляють гори, а темніші – долини або воду. Така карта вже має гарний вигляд, але її недостатньо і вона все ще далека від справжньої карти висот. Тому порогові значення та розмір карти можна змінювати програмно. Це дасть результати, необхідні для подальшої обробки та роботи.

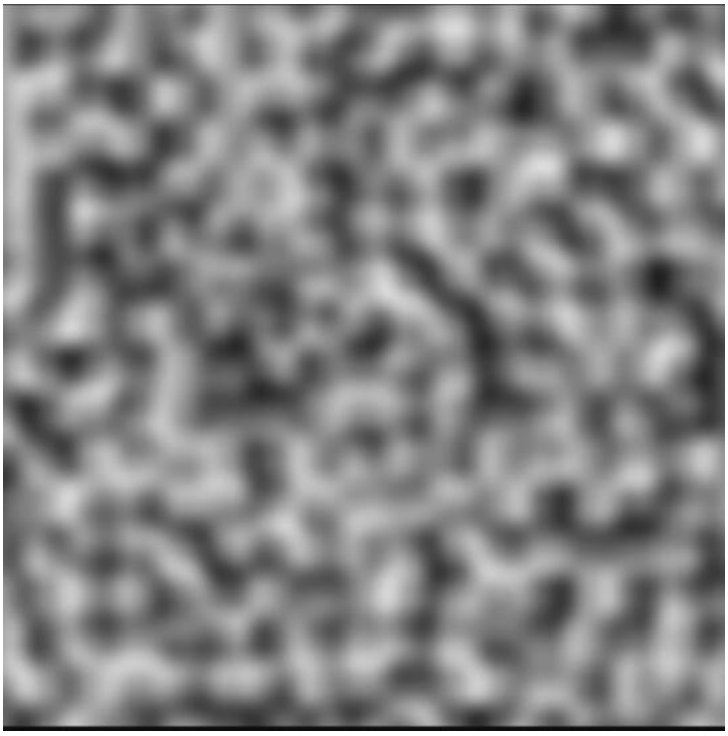

Рисунок 2.5 - Шум Перлина

Далі зображення перетворюється у висоти, тобто масив значень. Після цього ми отримаємо інформацію про висоту кожної ділянки карти. Залежно від висоти та вхідних умов, додається умови світу, такі як вода, біоми, клімат

тощо, і результатом цієї генерації є 2D або 3D світ, який можна використовувати далі. Параметри та фільтри для додавання елементів до кінцевої карти залежать від вхідних даних та внутрішнього алгоритму генерації (біоми, гори, річки та правила генерації облямівки).

Алгоритм генерує досить правдиву різноманітність світів і досить простий у використанні та модифікації. Він пропонує багато значень та аспектів для маніпуляцій та кастомізації.

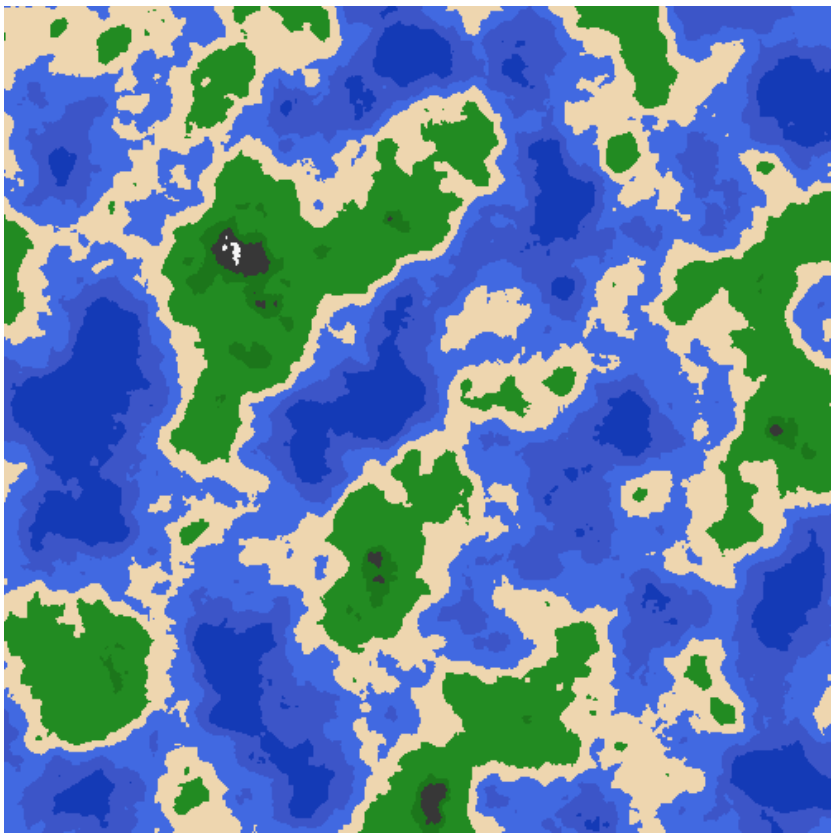

Рисунок 2.6 - Приклад генерації алгоритму

2.4 Клітинний автомат

Клітинний автомат - це дискретна математична модель, вперше використана в 1940 році. Незважаючи на складні завдання, які вирішує цей алгоритм, базова модель дуже проста.

По суті, клітинний автомат складається з N комірок і правил переходів між ними. Кожна комірка може перебувати в певному стані (вимкнена або увімкнена, як приклад найпростішого стану). Початковий розподіл комірок є основою автомата.

Кожен крок алгоритму одночасно переводить всі клітинки в новий стан за певними правилами. Сусідство клітинки визначає, які клітинки і як впливають на цей стан.

Правила зазвичай визначаються "правилами зміни стану". Це правило вказує, в якому стані перебуватиме клітина на наступному кроці, залежно від поточного стану клітини та стану її сусідів.

У клітинних автоматах існує поняття "покоління", яке відноситься до стану сітки клітин в даний момент часу. Починаючи зі стану першого покоління, за допомогою правил зміни стану можна обчислити стан наступного покоління, а потім стан після наступного покоління і так далі.

Оскільки ці процеси можуть повторюватися нескінченно, на клітинній сітці можна генерувати складні структури та патерни. Клітинні автомати використовуються в науці, інформатиці, фізиці, хімії та біології для моделювання різних фізичних і природних процесів, а також у промисловості для створення графіки та візуальних ефектів та ігор, ігрового контенту.

Щоб створити автомат, спочатку необхідно створити сітку і розмістити на ній клітини. Потім необхідно визначити стани, в яких може перебувати кожна клітинка.

Далі активується алгоритм, який генерує плитки. Він запускається кілька разів, і на кожному кроці плитки генеруються і підлаштовуються у вихідному просторі за правилами, заданими заздалегідь.

Рисунок 2.7 - Простий приклад роботи клітинному автомату

Наступним кроком є перетворення двовимірних карт у тривимірний простір, де плавні переходи між високими та низькими ділянками можуть бути побудовані або вбудовані в початковий автомат.

Цей алгоритм добре працює з невеликими ділянками мапи, печерами або природними лабіринтами. Розгортання генерації величезних 3D-просторів вимагає додаткових операцій на виході, але це цілком можливо.

2.5 Алгоритм фрактального шуму

Алгоритми генерації фрактального шуму створюють складні, деталізовані текстури шляхом додавання декількох окремих шумових шарів з різними параметрами. Кожен шар шуму створюється за допомогою шуму Перлина або симплекс-шуму - функції, яка генерує випадкові значення шуму, розподілені по поверхні.

На відміну від шуму Перлина, алгоритм фрактального шуму базується на фракталах.

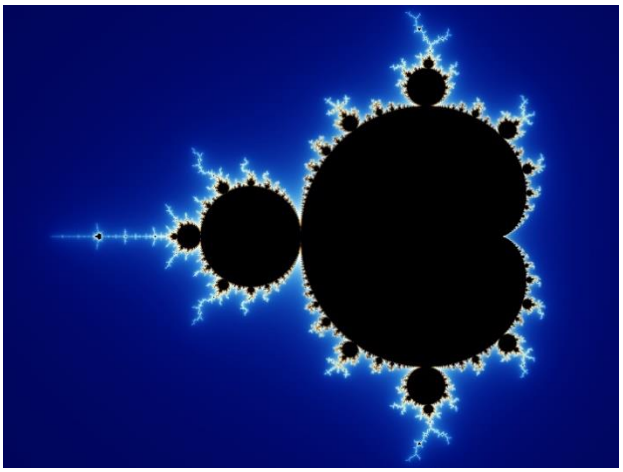

Рисунок 2.8 - Приклад фрактала

Щоб згенерувати світ за допомогою фрактального шуму, потрібно виконати наступні кроки:

1. Визначити вхідні параметри та їх значення

2. Для кожного пікселя зображення обчислити значення шуму шляхом додавання значень шуму окремих шарів (кожен шар шуму генерується за допомогою шуму Перлина або симплекс-шуму з відповідною частотою та амплітудою. Частота та амплітуда визначаються з значень шуму).

3. На основі отриманої текстури визначається тип ландшафту для кожної точки простору за допомогою вхідних параметрів.

Результат на виході є реалістичнішим у порівнянні з традиційною генерацією шуму Перлина, але сам алгоритм є більш складним у реалізації та налаштуванні.

2.6 Висновок до розділу 2

В цьому розділі було проаналізовано найпопулярніші алгоритми генерації 3Д світів та детальніше розібрано алгоритм їх роботи та реалізації. Було наведено приклади вихідних результатів роботи алгоритмів та сфери їх можливого використання.

РОЗДІЛ 3. Реалізація алгоритму генерування 3Д світу на основі вхідних даних користувача

3.1 Порівняння алгоритмів генерації

У минулому розділі було розглянуто обширний перелік найпопулярніших алгоритмів генерації 3Д світів у ігровій індустрії. Очевидно, що кожен з них має свої переваги та недоліки. В цьому розділі буде порівняно всі алгоритми та вибрано один для практичної реалізації та застосування.

Алгоритм Diamond Square – Метод генерації за допомогою рекурсії, на виході якої отримується карта висот. Алгоритм ділить карту на квадрати та ромби, після чого заповнює карту висот, де значення висоти обчислюється додаванням випадкової величини до середнього значення області на мапі.

Переваги:

- Алгоритм дозволяє генерувати реалістичні ландшафти з детальною структурою.
- Алгоритм генерує висотні карти з плавною зміною висот, що робить їх придатними для подальшого використання в іграх.

Недоліки:

- Алгоритм повільно працює для мап великих розмірів.
- В залежності від вхідних параметрів, можуть з’являтись помітні артефакти генерації.

Алгоритм згортання хвильової функції – метод генерації рівномірних рандомних світів за допомогою згортання хвильової функції за деяким вхідним шумом. Хвильова функція є фіксованою за періодом та формою, але шум може бути різний. Після виконання функції, отримується мапа з випадковою генерацією.

Переваги:

- Алгоритм дає можливість генерувати світи з різним рівнем деталізації.
- Може створювати світи з гладкою структурою.

Недоліки:

- Важкий для використання для мап великого розміру.
- Важкий для побудови складних структур.

Шум Перлина – алгоритм генерації реалістичних текстур, який використовує випадкові значення для створення коливань. Значення шуму обчислюється за допомогою певних функцій, а на виході ми отримуємо текстуру яку можна використовувати для генерації певної мапи.

Переваги:

- Дозволяє генерувати реалістичні карти висот.
- Генерує текстури зі складними та дрібними деталями.
- Можна використовувати для генерації текучих поверхонь.

Недоліки:

- Важкий у використанні для генерації складних форм.

- На високих рінях деталізації можливі артефакти.

Клітинний автомат – метод генерації текстур та форм, що базується на сітці (клітинах). Метод генерує карти з обмеженими областями. Кожна клітина має обмежений розмір та випадкове значення для створення текстур.

Переваги:

- Легкий у реалізації та використанні.
- Надає можливість для згладжування границь між клітинами для природнього вигляду.
- Можливість генерування різних форми з різними параметрами.

Недоліки:

- Обмеженість в використанні та генеруванні.
- Не дуже ефективний при генерації складних структур.
- Потребує додаткового оброблення та додавання деталей.

Алгоритм фрактального шуму – метод генерації шуму, що використовує математичні функції для створення структур. Також використовується для генерування високої якості текстур або деталей.

Переваги:

- Генерування складних форм з деталізацію на різних рівнях.
- Параметризація алгоритму.
- Паралельна робота для прискорення генерації.

Недоліки:

- При великих об’єктах генерація може бути повільним та ресурсоємким.
- Вимагає додаткового опрацювання після генерації.
- Є ризик створення артефактів в окремих частинах результату.

З перерахованих методів генерації був обраний алгоритм шуму Перлина, бо це потужний та ефективний метод генерації реалістичних світів. Алгоритм швидкий, простий в налаштуванні і забезпечує високу якість генерації.

Шум Перлина дозволяє налаштовувати багато параметрів, також його можна легко комбінувати з іншими методами генерації, для створення складних інтерактивних середовищ.

Таким чином, шум Перлина один з найпопулярніших й найкращих методів генерації світів. Через це він і був обраний для подальшого адаптування та реалізації в розробницькому середовищі.

3.2 Вибір технологій

В будь якому процесі розробки, вибір технологій та середовища розробки грає велику роль. Зараз на ринку є багато програм, застосунків та середовищ для створення і імплементації алгоритму генерації світів.

Оскільки генерація буде створена для можливої подальшої реалізації у ігровому застосунку, то не має необхідності у написанні рушія для програми з нуля або на базі певного SDK. На ринку зараз є багато ігрових рушіїв які

дозволять реалізувати алгоритм Перлина швидко і не турбуючись про графічну частину, рендер та роботу з інпутом.

Два основних застосунка, які зараз є актуальними на ринку розробки це Unity та Unreal Engine:

Unity - це платформа розробки ігор та інтерактивних додатків, що дозволяє створювати 2D та 3D ігри для різних платформ, таких як Windows, MacOS, Android, iOS, Xbox, PlayStation.

Unity має вбудований редактор, який дозволяє створювати об'єкти, надавати їм властивості, додавати компоненти та редагувати їх параметри. Редактор також підтримує інтуїтивний інтерфейс перетягування і розміщення елементів на сцені, що надає можливість швидко створювати та налаштовувати світи.

Unity підтримує багато мов програмування, таких як C#, JavaScript та Boo. Крім того, Unity має велику бібліотеку компонентів та ресурсів, що дозволяє розробникам легко створювати графічні та звукові ефекти, моделі, персонажів, анімації та інші елементи з використанням цих бібліотек.

Одним з головних переваг Unity слугує можливість одночасно розробляти під різні платформи. За допомогою Unity, розробники можуть створювати ігри одночасно на декілька ігрових платформ, що дозволяє їм заощадити час та зусилля при розробці та підтримці гри.

Unreal Engine - це ігровий рушій, розроблений компанією Epic Games. Він використовується для створення відеоігор, а також для візуалізації та

анімації в галузі архітектури, інтер'єру, автомобільної промисловості та інших галузях.

Unreal Engine має вбудований набір інструментів, які дозволяють створювати ігри різних жанрів та платформ, включаючи ПК, консолі, мобільні пристрої та віртуальну реальність. Він має потужний вбудований редактор, який дозволяє легко створювати і модифікувати ігрові об'єкти, додавати їм фізичні властивості, налаштовувати освітлення та матеріали.

Unreal Engine також має велику спільноту розробників, яка активно підтримується розробниками самого рушія. У цій спільноті зібрано багато корисних матеріалів, ресурсів та інструментів, що дозволяє швидко та ефективно розробляти ігри на основі Unreal Engine.

Однією з ключових особливостей Unreal Engine є можливість використання мови програмування C++. Це дозволяє розробникам створювати складні та потужні ігрові системи на основі низькорівневої мови програмування, що надає можливість розширювати рушій. Крім того, Unreal Engine підтримує багатопоточну обробку, що забезпечує оптимальну роботу з багатоядерними процесорами.

Оскільки Unity має більшу популярність серед розробників на ринку і є гнучкішим і простішим у використанні, то для реалізації алгоритму було обрано саму цей ігровий рушій. Мова розробки була обрана C#, бо вона надає всі переваги строго типізованої та об’єктно орієнтованої мови програмування, та є однією з основних мов для розробки на Unity.

3.3 Створення програмному додатку

В якості додаткового індивідуального завдання для кваліфікаційної роботи, мною був створений застосунок який генерує світ з використанням шуму Перлина та мапи падіння. Додаток був створений на оптимальних для поставленого завдання технологіях: Unity та мові C#. В додатку до кваліфікаційної роботи буде наданий код та файл-виконавець через який можна буде побачити фінальну реалізацію застосунка.

Для початку було створено основний скелет проекту та налаштовані змінні робочого середовища. Також були створені основні об’єкти на ігровій сцені та файли для подальшої роботи з проектом.

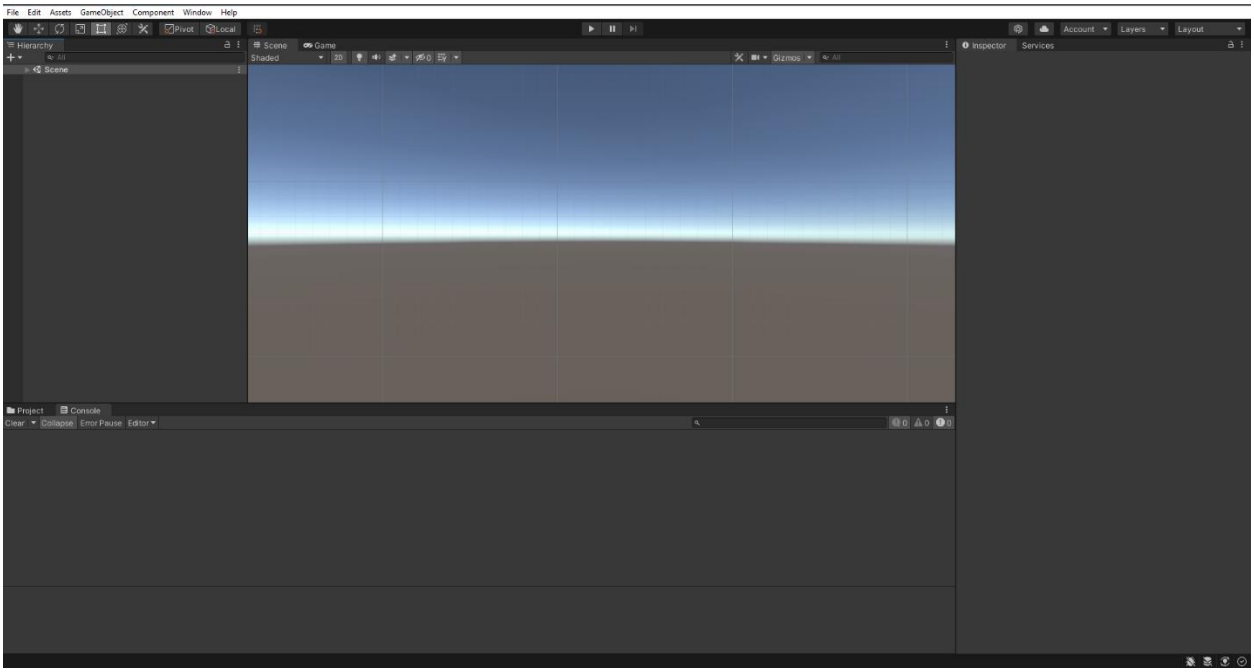

Рисунок 3.1 – Основне меню Unity.

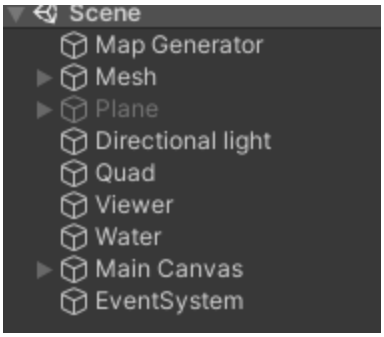

Рисунок 3.2 – Ієрархія об’єктів

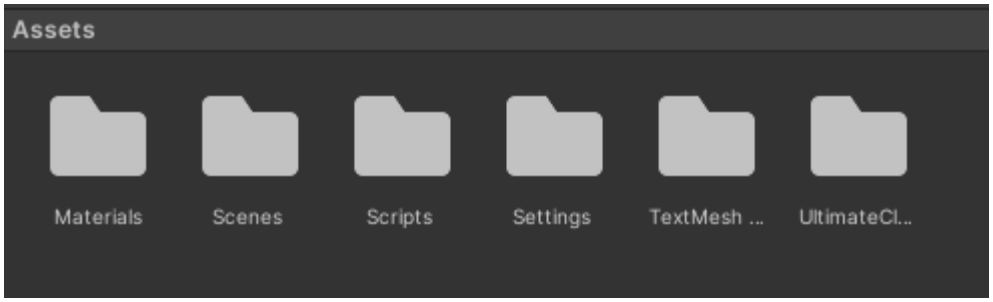

Рисунок 3.3 – Ієрархія файлів для подальшої роботи

3.4 Генерування мапи шуму

Після первинного налаштування, основною метою виступає створення мапи шуму. Для цього можна використати вбудований метод генерації двомірного масиву значень в мову C# та виконати деякі перетворення та покращення, після чого відмалювати результат на текстуру й накласти її на об’єкт.

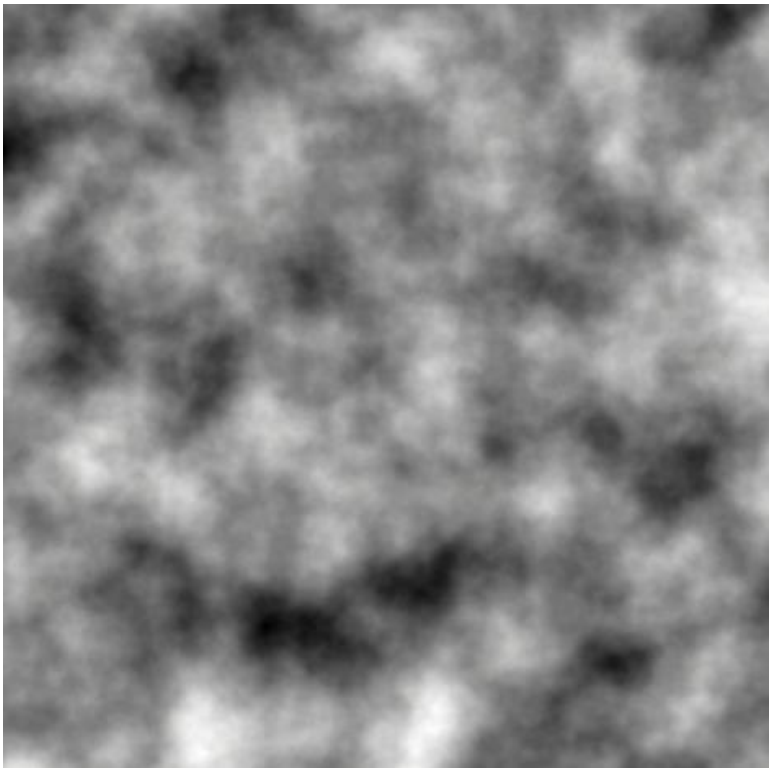

Рисунок 3.4 – Шум Перлина

Процес генерації та обробки шуму: Спочатку генерується двомісний масив типу float з висотою та шириною нашого шуму. Далі задаємо генератор і передаємо в нього сід – унікальне число для генерації. Основні параметри з якими буде працювати застосунок опрацьовуються саме тут. Після чого ініціалізується та обраховуємо параметри зсуву, максимальної висоти та щільності. Наступним кроком заповнюєтсья по одному всі значення за допомогою вбудованої бібліотеки Mathf та її методу PerlinNoise(float x, float y). Після заповнення масиву треба проітеруватися по кожній клітинці та вирахувати в залежності від позиції та параметрів нове значення. До кожного елементу враховується амплітуда, частота, висота шуму та зсув. Останнім кроком по генерації буде вирівнювання локальних висот і нормалізація методом зворотного лерпу (Inverse Lerp). Це необхідно

для того, щоб привести новоутворені значення до потрібних. В роботі за рівень глибокого моря рахувалися параметри близькі до 0, а за максимумі – 1. На виході створюється масив значень які можна перетворити в градієнт сірого і вивести на меш або об’єкт, який і буде базою для генерації. Для створення текстурі градації сірого, можна для кожного значення пропорцією вирахувати значення кольору між чорним та білим.

3.5 Генерація мапи світу

На цьому етапі вже створений шум, який буде слугувати картою висот, для генерації об’єму та висот майбутнього простору. Шум Перлина для кожної точки містить значення від 0 до 1, де 0 це абсолютний мінімум, або в нашому випадку рівень моря, а 1 це найвища точка в згенерованому світі. Зробивши не складні перетворення, для кожного значення висоти задаємо значення кольору, з цих значень утворюємо карту кольорів, в майбутньому нашу текстуру, і перевіряємо її вимальовку. Значення кольорів задані заздалегідь. 0 це глибока вода, 0.1-0.15 не глибокі озера та пляжі, а далі все залежить від клімату але у цьому випадку пляжі, трава, ліси, гори і в кінці вершини гір зі снігом.

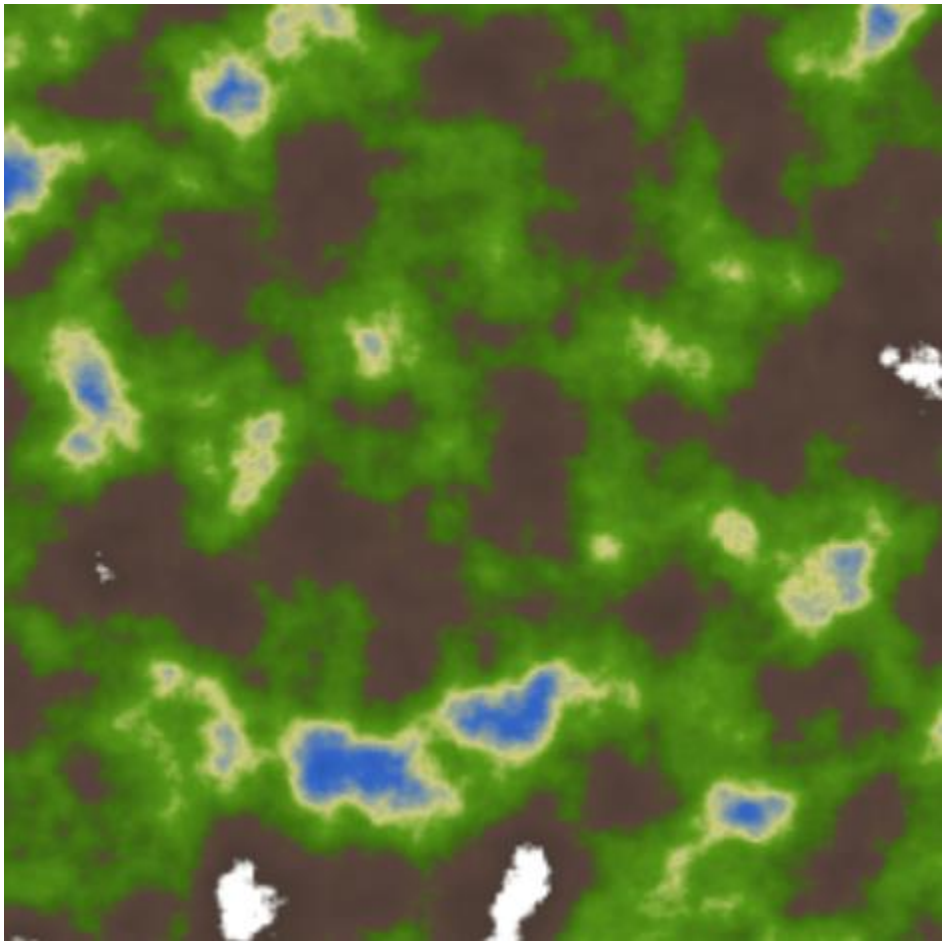

Рисунок 3.5 – Мапа кольорів

Наступним кроком буде створення мешу, об’єкту на який буде виступати твердою поверхнею нашого ландшафту. Для цього знову беремо карту висот (шум Перлина) та генеруємо сорозмірний меш, який ділимо на довільну кількість полігонів – чим більше, тим деталізованім буде остаточний результат. Для ділення можна використати схожий принцип що і в алгоритмі Diamond-Square. Поверх меша додаємо нашу мапу кольорів, як текстуру, і отримуймо результат наближений до фінального. Для створення текстури яку можна використовувати з мешем, потрібно ще створити

матеріал, але оскільки в матеріал не потребує інших параметрів, то однієї текстури буде досить.

Далі треба вирахувати значення висоти по осі Y кожного полігона. Для цього треба знайти середнє значення змінних в масиві шуму та вирахувати приблизну висоту вертекса, після чого змінити його положення. Висота та положення вертекса вираховується за спеціальною кривою, для можливості гнучкого редагування, але не перевищую значення максимальної вистоти заданої заздалегідь.

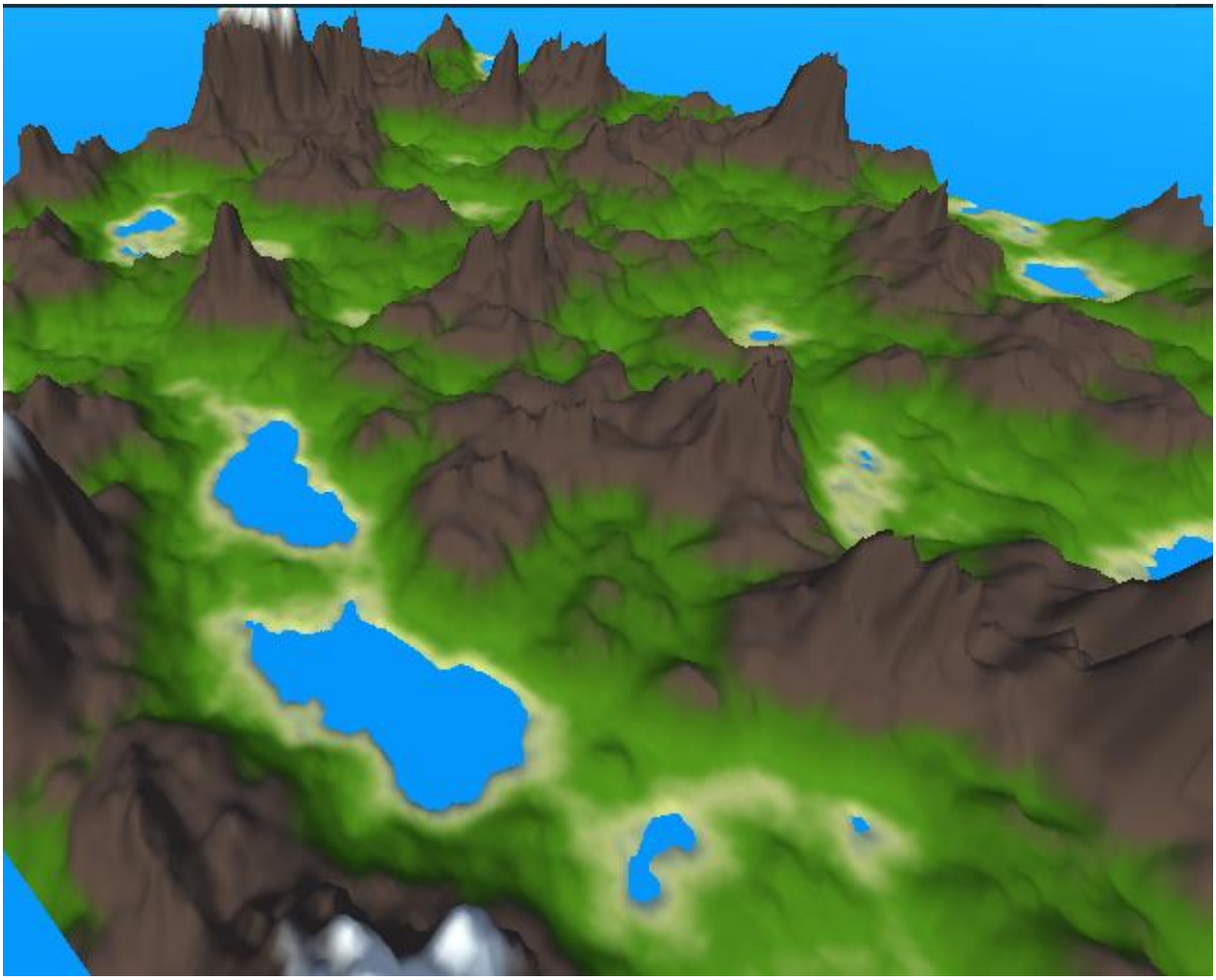

Рисунок 3.6 – Згенерований меш

На даному етапі ми отримали непоганий результат, але ігрова мапа занадто різко обривається на краях. Для вирішення даного питання можна використати карту падіння – карта за якою ми будемо згладжувати кінці мапи до рівня моря.

Метод генерації карти падіння досить простий, треба просто згладжувати краї нашого шуму, а точніше накласти на нього ще одну мапу, де 1 – це початкова висота, а 0 це рівень моря. Після генерації ми отримаємо чорний квадрат який уходить плавно в білий колір.

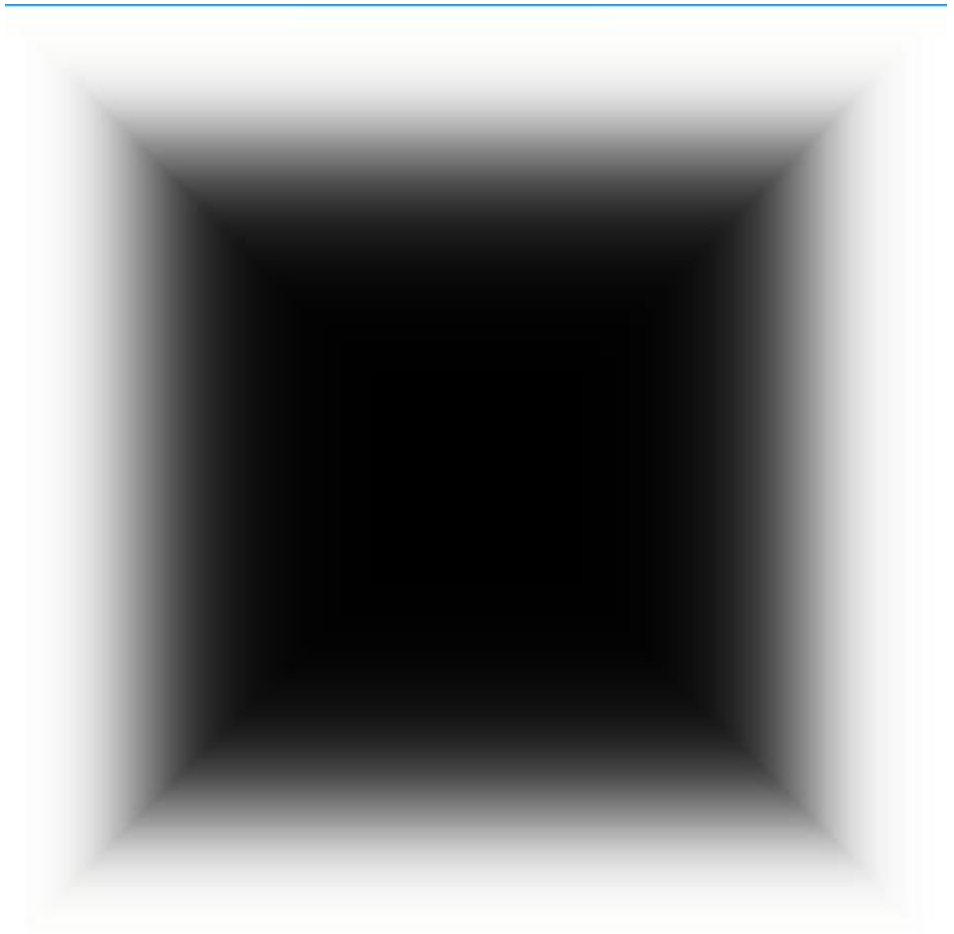

Рисунок 3.7 – Карта падіння

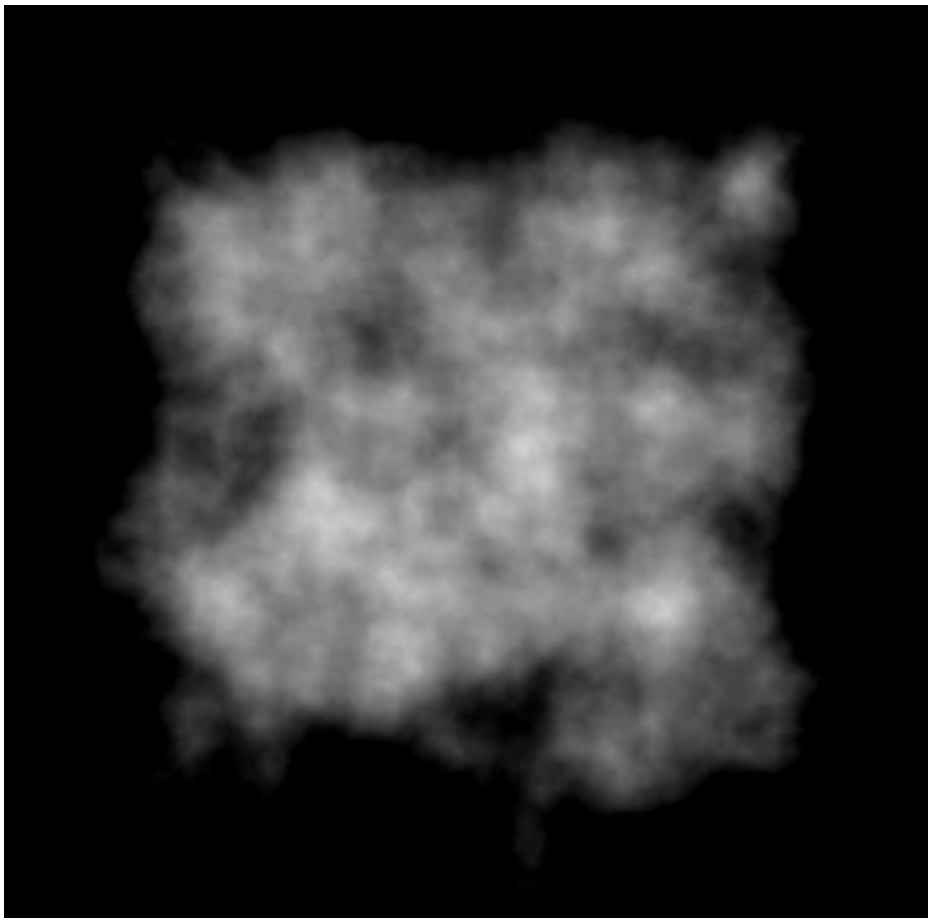

Рисунок 3.8 – Шум Перлина з картою падіння

Об’єднавши всі попередні кроки, на виході ми отримаємо текстуру шуму, яка буду слугувати нашою основою для подальшої модифікації світу. Об’єднання відбувалося методом додавання та заокруглення значень до [0,1], за схожим методом, що був використаний раніше. Сама мапа схожа на острів або материк.

3.6 Отримання результату

Здійснивши всі кроки з минулого підрозділу ми отримаємо фінальний результат нашої генерації та імплементації алгоритму. Також вхідні параметри алгоритму можна змінювати задля отримання бажаного та варіативного результату. Застосунок надає такі параметри для налаштування алгоритму:

Draw Mode –Тип вимальовування мапи. Має 4 параметри: шум Перлина, карта кольорів, карта падіння та фінальний меш.

Noise Scale – визначає розмір шуму Перлина, що буде використаний для генерації. Більше значення дасть більш хаотичний результат на малій мапі.

Octaves - параметр для регулювання окремих елементів згенерованого ландшафту

Persistance - наскільки високий поріг для вирішення типу місцевості

Lacunarity - наскільки хаотично спотворення мапи шуму буде зроблено перед генерацією

Seed – зерно для генератора псевдорандому. Шум генерується на основі цього значення.

Offset X – зсув по осі X.

Offset Y – зсув по осі Y.

Use Falloff – чи використовувати карту падіння.

Mesh Height – Максимальна висота майбутнього мешу.

Рисунок 3.9 – Вигляд застосунку

3.7 Висновок до розділу 3

В цьому розділі було розглянуто та порівняно всі описані раніше алгоритми генерації 3Д світів в ігровій індустрії. Було реалізовано та пояснено реалізацію обраного алгоритму – шуму Перлина. Було розібрано нюанси розробки та можливості контролю роботи алгоритму та створено демонстративну програму.

ВИСНОВОК

Розвиток алгоритмів генерації віртуальних світів для ігрової індустрії почався давно. Розробники винаходили багато різних методів для вирішення проблем генерації у своїх проектах. На сьогодні існує велика кількість алгоритмів для вирішення будь якої проблеми генерації реалістичного та детального світу.

В цій роботі було розглянуто поняття віртуальний світ, проблеми, що вирішує процедурна генерація та як можна використати процедурну генерацію для створення нескінченної кількості унікального ігрового контенту.

Було розглянуто основні алгоритми процедурної генерації 3Д світів серед яких були: Алгоритм Diamond-Square, згортання хвильової функції, шум Перлина, клітинний автомат та алгоритм фрактального шуму.

Всі вищеназвані алгоритми були порівняні і виписані переваги і недоліки кожного з методів генерації 3Д світів.

В додаток до роботи, було розроблено і продемонстровано процес розробки застосунку, що використовує шум Перлина для генерації світу. Застосунок також надає можливість редагувати вхідні данні. В додаток до роботи надано вихідні коди та файл для запуска застосунку.

ВИКОРИСТАНІ ДЖЕРЕЛА

[1] [Стаття] “Defining Virtual Worlds and Virtual Environments” автора Ральфа Шроедара

Режим доступу до ресурса:

https://jvwr-ojs-utexas.tdl.org/jvwr/article/view/294

[2] [Стаття] “Toward a Definition of “Virtual Worlds” автора Марка Белла

Режим доступу до ресурса:

https://www.semanticscholar.org/paper/Toward-a-Definition-of-%E2%80%9CVirtual-Worlds%E2%80%9D-Bell/56f9ea62a05da69d1a37dcbf3c4d8e324ad73794

[3] [Книга] “Moving beyond the game: Social virtual worlds” авторки Бітсі Бук

[Книга] "Procedural Content Generation in Games" авторів Ноеля Ларчера та Тьєррі Деріче

[Книга] "Generative Art: A Practical Guide Using Processing" автора Метью Пламмера

[Книга] "Procedural Generation in Game Design" авторів Тані Ковач та Тілера Ходгінсона

[Книга] "The Nature of Code" автора Деніеля Шифмана

[Книга] "Artificial Intelligence for Games" автора Іана Міллінгтона

[Стаття] "Introduction to Procedural Generation" автора Аміта Паріха

[Стаття] "Procedural Generation: The Future of Game Development?" автора Деніела Бернхарда

[Стаття] "A Beginner's Guide to Procedural Content Generation" автора Ендрю Джонсона

[Стаття] "Procedural Generation of Game Content: A Survey" авторів Ноелі Ларчера, Тьєррі Деріче та Джорджіо Карфагно

[Стаття] "Procedural Generation: How It Works and Why It Matters" автора Мікала Губенцева

[Ресурс інтернету] Procedural world generation

Режим доступу до ресурса:

https://www.mit.edu/~jessicav/6.S198/Blog_Post/ProceduralGeneration.html

[Ресурс інтернету] How to create Infinite Worlds in Your Games Using Procedural Generation

Режим доступу до ресурса:

https://betterprogramming.pub/how-to-create-infinite-worlds-in-your-games-using-procedural-generation-d4d4baecd3cb

[Ресурс інтернету] Procedural Generation – A Comprehensive Guide Put in Simple Words

Режим доступу до ресурса:

https://scaleyourapp.com/procedural-generation-a-comprehensive-guide-in-simple-words/

[Ресурс інтернету] Procedural Terrain Generation: Diamond-Square

Режим доступу до ресурса:

http://jmecom.github.io/blog/2015/diamond-square/

[Ресурс інтернету] Diamond Square Terrain Generation

Режим доступу до ресурса:

https://www.ianhunter.ie/posts/diamond-square-terrain-generation/

[Ресурс інтернету] Generating Worlds With Wave Function Collapse

Режим доступу до ресурса:

https://www.procjam.com/tutorials/wfc/

[Ресурс інтернету] Procedural generation using Wave Function Collapse

Режим доступу до ресурса:

https://applaudostudios.com/blog/posts/procedural-generation-using-wave-function-collapse

[Ресурс інтернету] Making maps with noise functions

Режим доступу до ресурса:

https://www.redblobgames.com/maps/terrain-from-noise/

[Ресурс інтернету] Generation Digital Worlds Using Perlin Noise

Режим доступу до ресурса:

https://medium.com/nerd-for-tech/generating-digital-worlds-using-perlin-noise-5d11237c29e9

[Ресурс інтернету] Generation Terrain with Cellular Automata

Режим доступу до ресурса:

https://medium.com/universe-factory/this-post-is-about-a-world-building-tool-ive-been-working-on-5d9560844ff5

[Ресурс інтернету] Procedural Level Generation in Games Using a Cellular Automaton

Режим доступу до ресурса:

https://www.kodeco.com/2425-procedural-level-generation-in-games-using-a-cellular-automaton-part-1

[Ресурс інтернету] Creating procedural Mountains: A Fractal Noise tutorial

Режим доступу до ресурса:

https://devforum.roblox.com/t/creating-procedural-mountains-a-fractal-noise-tutorial/1494362

[Ресурс інтернету] Techniques from Fractal Terrain Generation

Режим доступу до ресурса:

https://web.williams.edu/Mathematics/sjmiller/public_html/hudson/Dickerson_Terrain.pdf

ВИКОРИСТАНІ АССЕТИ

Ultimate clean GUI Pack by **gamevanilla**

Режим доступу до ресурса:

https://assetstore.unity.com/packages/2d/gui/ultimate-clean-gui-pack-154574

https://www.gamevanilla.com/