Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
(повна назва)

Кафедра *Системотехніки*
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти *другий (магістерський)*

Дослідження та аналіз алгоритмів процедурної генерації карт у 2D іграх
(тема)

Виконав:
студент *II* курсу, групи *СПРм-22-2*
Балюк І.В.
(прізвище, ініціали)

Спеціальність *122 Комп'ютерні науки*
(код і повна назва спеціальності)

Тип програми *освітньо-наукова*
(освітньо-професійна або освітньо-наукова)

Освітня програма *Системне проектування*
(повна назва освітньої програми)

Керівник *проф. Іванов В. Г.*
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ *Гребеннік І. В.*
(підпис) (прізвище, ініціали)

2024 р.

Я, як студент ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Кваліфікаційна робота не містить відомостей заборонених до відкритого опублікування.

Кваліфікаційна робота виконана у відповідності до стандартів, що діють в Україні.

Попередній захист проведено «15» червня 2024 р.

Керівник кваліфікаційної роботи проф. Іванов В.Г

Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*

Кафедра *Системотехніки*

Рівень вищої освіти *другий (магістерський)*

Спеціальність *122 Комп'ютерні науки*
(код і повна назва)

Тип програми *освітньо-наукова*
(освітньо-професійна або освітньо-наукова)

Освітня програма *Системне проектування*
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«____»_____ 20 ___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові *Балюку Іллі Васильовичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *«Дослідження та аналіз алгоритмів процедурної генерації карт у 2D іграх»*
затверджена наказом університету від *«01» квітня 2024 р.* № *259 Ст*
2. Термін подання студентом роботи до екзаменаційної комісії *«18» червня 2024 р.*
3. Вихідні дані до роботи *Об'єкт дослідження – процес процедурної генерації карт у 2D іграх. Предмет дослідження – алгоритми процедурної генерації карт у 2D іграх. Предмет розробки – додаток, що реалізує алгоритми процедурної генерації карт у 2D іграх. Технічне забезпечення: IBM-сумісний персональний комп'ютер. Перелік використовуваних програмних засобів: ОС Microsoft Windows 10, мова програмування C#, інтегроване середовище розробки Unity.*
4. Перелік питань, що потрібно опрацювати в роботі *4.1 Перелік скорочень і термінів. 4.2 Вступ. 4.3 Аналіз предметної області. 4.4 Постановка задачі. 4.5 Призначення розробки. 4.6 Мета проведення дослідження. 4.7 Існуючі підходи та їх актуальність. 4.8 Складова методів генерації 2D карт місцевості. 4.9 Опис методів генерації. 4.10 Аналіз складових методів генерації. 4.11 Проведення дослідження. 4.12 Методологія проведення експерименту. 4.13 Обґрунтування середовища розробки. 4.14 Специфікація програмного забезпечення. 4.15 Реалізація методів генерації. 4.16 Реалізація алгоритму шум перліна (Perlin noise). 4.17 Реалізація алгоритму направлений тунель (Directional tunnel). 4.18 Реалізація алгоритму клітинний автомат (Cellular Automata). 4.19 Реалізація алгоритму випадкового блукання (Random Walk). 4.20 Розробка демонстраційного додатку. 4.21 Аналіз результатів. 4.22 Вибір критеріїв для оцінки методів генерації. 4.23 Оцінка і аналіз результатів. 4.24 Висновок.*

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п. 5 включається до завдання за рішенням випускової кафедри) *5.1 Результати генерації методом шум перліна (Perlin noise). 5.2 Результати генерації методом направлений тунель (Directional tunnel). 5.3 Результати генерації методом клітинний автомат (Cellular Automata). 5.4 Результати генерації методом випадкового блукання (Random Walk). 5.5*

Результати роботи демонстраційного додатку.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Основна частина	*Проф. Іванов В.Г.*		

КАЛЕНДАРНИЙ ПЛАН

	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
	Отримання та аналіз завдання, уточнення плану роботи	*11.03.24*	виконано
	Аналіз методів генерації 2D карт ландшафіу	*13.05.24*	виконано
	Огляд існуючих методів генерації 2D карт ландшафіу	*15.05.24*	виконано
	Розробка вимог до розробки додатку	*18.05.24*	виконано
	Розробка програмного забезпечення задачі	*24.05.24*	виконано
	Проведення експериментальних досліджень	*28.05.24*	виконано
	Оформлення пояснювальної записки	*31.05.24*	виконано
	Підготовка презентації	*01.06.24*	виконано
	Подання закінченої роботи науковому керівникові	*03.06.24*	виконано
0	*Подання роботи на рецензування*	*10.06.24*	виконано
1	*Попередній захист*	*15.06.24*	виконано
2	*Подання роботи до екзаменаційної комісії*	*17.06.24*	виконано

Дата видачі завдання *«01» квітня 2024 р.*

Студент _____
(підпис)

Керівник роботи _____ *проф. Іванов В. Г.*
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Звіт з передатестаційної практики: 66 с., 14 рис., 1 табл., 25 джерел інформації. ПРОЦЕДУРНА ГЕНЕРАЦІЯ КАРТ, DATA PIPELINE, 2D, UNITY, C#.

Об’єктом дослідження є алгоритми процедурної генерації карт у 2D іграх.

Предметом досліджень є особливості та ефективність алгоритмів процедурної генерації карт у контексті 2D ігор.

Мета роботи - дослідити та проаналізувати алгоритми процедурної генерації карт у 2D іграх з метою виявлення їхніх переваг та недоліків.

Методи дослідження – використання Unity для розробки і тестування різних алгоритмів процедурної генерації карт, аналіз коду та результатів генерації.

Сфера застосування – розробка комп'ютерних ігор, віртуальна реальність, навчальні програми з комп'ютерної графіки та програмування.

ABSTRACT

Report on pre-certification practice: 66 pages, 14 pictures, 1 table, 25 sources of information. PROCEDURAL CARD GENERATION, DATA PIPELINE, 2D, UNITY, C#.

The object of research is algorithms for procedural map generation in 2D games.

The subject of research is the peculiarities and effectiveness of procedural map generation algorithms in the context of 2D games.

The purpose of the work is to investigate and analyze algorithms for procedural map generation in 2D games in order to identify their advantages and disadvantages.

The research methods are the use of Unity for the development and testing of various algorithms for procedural map generation, code analysis and generation results.

The field of application is the development of computer games, virtual reality, educational programs in computer graphics and programming.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ 9

ВСТУП 10

1 ПРОВЕДЕННЯ АНАЛІЗУ ПРЕДМЕТНОЇ ОБЛАСТІ 12

1.1 Аналіз предметної області 12

1.2 Постановка задачі 13

1.2.1 Призначення розробки 14

1.2.2 Мета проведення дослідження 14

1.3 Існуючі підходи та їх актуальність 15

2 СКЛАДАЮЧА МЕТОДІВ ГЕНЕРАЦІЇ 2D КАРТ МІСЦЕВОСТІ 18

2.1 Опис методів генерації 18

2.2 Аналіз складових методів генерації 18

2.2.1 Шум Перліна (Perlin noise) 19

2.2.2 Направлений тунель (Directional tunnel) 22

2.2.3 Клітинний автомат (Cellular Automata) 24

2.2.4 Випадкове блукання (Random Walk) 28

3 ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ 30

3.1 Методологія проведення експерименту 30

3.2 Обгрунтування середовища розробки 31

3.3 Специфікація програмного забезпечення 32

3.4 Реалізація методів генерації 33

3.4.1 Реалізація алгоритму шум перліна (Perlin noise) 34

3.4.2 Реалізація алгоритму направлений тунель (Directional tunnel) 37

3.4.3 Реалізація алгоритму клітинний автомат (Cellular Automata) 39
3.4.4 Реалізація алгоритму випадкового блукання (Random Walk) 44
3.5 Розробка демонстраційного додатку 47
4 АНАЛІЗ РЕЗУЛЬТАТІВ 57
4.1 Вибір критеріїв для оцінки методів генерації 57
4.2 Оцінка і аналіз результатів 58
ВИСНОВОК 64
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ ТА ДЖЕРЕЛ ПОСИЛАННЯ 65
Додаток А 67
Додаток Б 79
Додаток В 95

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

2D - Two dimensional

PG - Procedural generation

ПГК – Процедурна генерація карт

C# - мова програмвання

Unity – середа розробки додатків

ВСТУП

У сучасному світі відеоігор, що розвивається на захопливій перетині технологій та креативності, процедурна генерація карт у 2D іграх набуває все більшого значення. Цей аспект гри стає ключовим у визначенні геймплейного досвіду та надає ігровим універсумам безмежність інтриги та різноманіття. Поглиблене дослідження та аналіз алгоритмів процедурної генерації карт у 2D іграх стає необхідним кроком для тих, хто мріє про новаторські геймплейні рішення та створення неповторних віртуальних світів.

Ця робота присвячена вивченню та аналізу різноманітних підходів до процедурної генерації карт у 2D іграх, з метою розкриття їхнього потенціалу, обмежень та можливостей для подальшого використання у практиці розробки відеоігор. Об'єктом дослідження є алгоритми, які відповідають за створення віртуальних карт, їх структуру, топологію та елементи взаємодії.

Глибоке розуміння та аналіз алгоритмів процедурної генерації карт у 2D іграх є ключовим для досягнення кращих результатів у розробці відеоігор, які здатні захопити увагу та залишити незабутні враження у гравців. Ця робота прагне відкрити нові горизонти у галузі геймдеву, дозволяючи розробникам ефективніше використовувати ресурси та зосередитися на створенні найбільш інноваційних та захоплюючих віртуальних світів.

У цьому контексті, робота пропонує вивчити та оцінити різноманітні алгоритми процедурної генерації карт у 2D іграх, а також зосередитися на їхніх можливостях та обмеженнях. Результати дослідження можуть стати важливим внеском у розвиток галузі та сприяти подальшій еволюції ігрової індустрії, надаючи нові можливості для творчості та інновацій в галузі розробки відеоігор.

Процедурна генерація карт дозволяє створювати ігрові універсуми безмежної просторової та сюжетної глибини, в яких гравці можуть відчути себе частиною чогось більшого, ніж просто гра. Вона відкриває нові можливості для

інтерактивного спілкування з ігровим оточенням та розвитку унікальних стратегій геймплею.

Розробники відчувають необхідність досліджувати різні методи та алгоритми процедурної генерації карт, адже від їхнього вибору залежить не лише зовнішній вигляд ігрового світу, але й його структура та геймплейні можливості. Тому глибоке розуміння цих процесів та їх впливу на геймплей є критичним для розробки інноваційних та успішних ігор.

Зокрема, використання Unity як платформи для реалізації та тестування різноманітних алгоритмів процедурної генерації дозволяє розробникам ефективно експериментувати з різними підходами та вибирати найбільш підходящі для свого проєкту. Такий підхід дозволяє прискорити процес розробки та забезпечити більш ефективне використання ресурсів.

Основна мета роботи полягає в розгляді та аналізі різних методів генерації карт місцевості, їхньої ефективності, складності в реалізації та потенційних переваг для ігрового досвіду. Використання Unity дозволяє не лише ефективно реалізувати різноманітні алгоритми, а й досліджувати їхні взаємозв'язки та можливості застосування у практичній галузі розробки ігор.

Це дослідження є актуальним та перспективним, оскільки розвиток процедурної генерації карт місцевості відкриває нові горизонти для створення цікавих, непередбачуваних та захоплюючих ігрових світів. Результати роботи можуть мати значущий вплив на подальший розвиток галузі розробки відеоігор та сприяти зростанню якості ігрового досвіду для мільйонів гравців по всьому світу.

1 ПРОВЕДЕННЯ АНАЛІЗУ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Процедурна генерація карт у 2D іграх є одним із ключових аспектів, який впливає на геймплей, реінгеймплей та загальний враження від гри. Пошук оптимальних алгоритмів для генерації карт виявляється важливим завданням для розробників, оскільки від цього залежить якість та цікавість ігрового досвіду.

Одним з напрямів у вивченні процедурної генерації карт є аналіз та вдосконалення різних алгоритмів, таких як алгоритми решітки, клітинних автоматів, шуму Перліна та генетичні алгоритми. Кожен з цих підходів має свої переваги та недоліки, і вибір конкретного методу залежить від потреб і особливостей конкретного проекту.

Аналіз сучасних досліджень та практичних застосувань процедурної генерації карт у 2D іграх вказує на те, що ця технологія широко використовується в різних жанрах, від рольових ігор до інді-проектів. Розвиток комп'ютерних технологій та зростання потужності обчислювальних пристроїв дозволяє розробникам створювати все більш складні та реалістичні ігрові світи за допомогою процедурної генерації.

Однак, разом з потужністю цих інструментів виникають і виклики, пов'язані з оптимізацією та підтримкою різних пристроїв. Ефективність алгоритмів генерації карт може визначати не лише якість гри, а й її роботу на різних платформах, включаючи мобільні та вбудовані системи.

Таким чином, вивчення та аналіз алгоритмів процедурної генерації карт у 2D іграх є актуальним завданням, яке може сприяти розвитку якісних та захоплюючих ігрових досвідів для гравців усього світу. Цей аналіз допоможе визначити найефективніші та найбільш підходящі алгоритми для різних типів

ігор та платформ, що дозволить розробникам максимально використовувати потенціал процедурної генерації для створення неповторних ігрових світів.

1.2 Постановка задачі

Метою даної магістерської роботи є дослідження та аналіз алгоритмів процедурної генерації карт у 2D іграх з використанням платформи Unity. Конкретні методи, що будуть досліджені, включають шум Перліна (Perlin noise), направлений тунель (Directional tunnel), клітинний автомат (Cellular Automata), околиця Мура (Moore Neighbourhood), околиця фон Неймана (von Neumann Neighbourhood) та випадкове блукання (Random Walk).

Для досягнення поставленої мети передбачено наступні завдання:

– провести детальний аналіз кожного з вищезазначених методів генерації карт, визначити їх особливості, переваги та недоліки;

– створити тестові середи для кожного з методів генерації карт у середовищі Unity, що дозволить візуально оцінити результати їх роботи та порівняти ефективність;

– провести експериментальне тестування кожного з методів генерації карт з використанням розроблених тестових середовищ. Зібрати дані про ефективність, реалізм та затрати ресурсів кожного методу;

– побудувати невеликий додаток у середовищі Unity, що демонструватиме роботу кожного з досліджених методів генерації карт в комбінації, показуючи їх можливість створення різноманітних ігрових світів;

– зібрати та проаналізувати отримані результати тестування кожного методу генерації карт. Визначити переваги та недоліки кожного підходу, а також їх придатність для конкретних ігрових проектів.

Результатом роботи буде детальний аналіз та порівняння різних методів процедурної генерації карт у 2D іграх, розроблені тестові середи для кожного з методів та невеликий додаток, що демонструє їх роботу в комбінації. Це

дозволить краще зрозуміти ефективність та придатність кожного методу для створення різноманітних ігрових світів у майбутніх проектах.

1.2.1 Призначення розробки

Призначення даної роботи полягає у отриманні практичних результатів використання різних методів процедурної генерації карт у 2D іграх та їх подальшому аналізі. Це дозволить:

– встановити, які методи генерації карт найбільш ефективні з точки зору якості, реалізму та оптимального використання ресурсів;

– виявити переваги та недоліки кожного з досліджених методів генерації карт, щоб надалі використовувати їх у майбутніх ігрових проектах з максимальною ефективністю;

– надати рекомендації та кращі практики щодо використання методів генерації карт у 2D іграх для підвищення якості та реалізму ігрових світів;

– внести свій внесок у розвиток галузі процедурної генерації карт у відеоіграх, надаючи нові інсайти та можливості для подальших досліджень та вдосконалення методів.

1.2.2 Мета проведення дослідження

Метою даної магістерської роботи є глибоке дослідження та аналіз алгоритмів процедурної генерації карт у 2D іграх з використанням різних методів, таких як шум Перліна (Perlin noise), направлений тунель (Directional tunnel), клітинний автомат (Cellular Automata), околиця Мура (Moore Neighbourhood), околиця фон Неймана (von Neumann Neighbourhood) та випадкове блукання (Random Walk).

Конкретні цілі дослідження включають:

– оцінку якості та реалізму карт, створених за допомогою кожного з вищезазначених методів, з метою визначення їхньої придатності для різних типів ігрових проектів;

– визначення які методи генерації карт є найбільш ефективними з точки зору реалізму, ресурсоємності та швидкодії;

– встановлення переваг і недоліків кожного методу генерації карт з метою надання рекомендацій розробникам ігор щодо їх використання;

– розробку невеликого додатку, який демонструватиме роботу кожного з досліджених методів генерації карт у комбінації, щоб показати їхні можливості та обмеження.

Отримання цих результатів сприятиме розумінню та вдосконаленню методів процедурної генерації карт у 2D іграх, що, в свою чергу, покращить якість ігрового досвіду для гравців та надасть нові можливості для розробників.

1.3 Існуючі підходи та їх актуальність

У галузі генерації ігрових світів використовуються як процедурні, так і не процедурні підходи. Далі буде порівняно такі методи, а саме їх застосування, випадки використання та конкретні приклади.

Процедурна генерація ігрових світів використовується в ігровій індустрії для створення великих, різноманітних ігрових світів з мінімальними ресурсами. Вона знайшла застосування у таких жанрах, як рольові ігри, платформери, стратегії та інші, де кожне проходження гри має бути унікальним.

У такого підходу можна виділити наступні переваги:

– кожне нове генерування світу дає унікальний ігровий досвід, що забезпечує переіграбельність;

– використання алгоритмів генерації дозволяє створювати великі світи з мінімальними обчислювальними ресурсами;

– процедурна генерація дозволяє легко розширювати розмір ігрового світу, не збільшуючи величезну кількість ручно розроблених ресурсів.

Також він має і певні недоліки:

– іноді процедурно згенерований світ може виглядати випадковим або нелогічним, що може знизити іммерсію гравця;

– розробники можуть втратити контроль над дизайном деяких аспектів гри, що може призвести до неочікуваних проблем або нестабільності геймплею;

– процедурно згенеровані світи можуть вимагати постійного підтримування та оновлення, оскільки алгоритми генерації можуть мати свої недоліки або вимагати коригування.

До прикладів застосування можна віднести наступні ігри: Minecraft - У цій грі ігровий світ генерується процедурно, що означає, що кожне нове світове створення дає унікальний ландшафт та структури. Spelunky - В цій платформеній грі кожен рівень генерується процедурно, що забезпечує невичерпну різноманітність та виклик.

Не процедурна генерація ігрових світів частіше використовується в іграх, де розробники хочуть точного контролю над дизайном ігрового світу, або в тих випадках, коли ігровий світ потребує великої уваги до деталей.

У такого підходу можна виділити наступні переваги:

– розробники мають повний контроль над кожним аспектом ігрового світу, що дозволяє створювати більш деталізовані та атмосферні локації;

– ручне створення світу дозволяє уникнути можливих проблем, пов'язаних з процедурною генерацією, таких як нестабільність або нелогічність;

– розробники можуть застосовувати свій художній стиль та виражати свої унікальні ідеї без обмежень, які можуть виникнути при процедурній генерації.

Також він має і певні недоліки:

– створення ігрового світу в ручну може бути часомістким та вимагати значних людських і фінансових ресурсів;

– створений світ в ручну може бути обмеженим за розмірами у порівнянні з тим, що може бути згенеровано процедурно;

– відсутність унікальності при кожному новому запуску гри може знизити реіграбельність та інтерес гравця.

До прикладів застосування можна віднести наступні ігри The Legend of Zelda: Breath of the Wild - в цій грі ігровий світ ретельно розроблений розробниками і не генерується процедурно. Кожна локація має свою унікальну архітектуру та дизайн. Ori and the Blind Forest: У цій метроїдванії ігрові рівні ретельно розроблені вручну, щоб забезпечити відчуття магічного світу.

Можна зазначити, що обидва підходи до створення ігрових світів, як процедурна генерація, так і не процедурна генерація, мають свої унікальні переваги та недоліки. Процедурна генерація дозволяє швидко та ефективно створювати унікальні, різноманітні світи з мінімальними ресурсами, забезпечуючи при цьому велику переіграбельність та розширення масштабів ігрового простору. Не процедурна генерація, натомість, дозволяє розробникам максимально контролювати кожен аспект ігрового світу, створюючи деталізовані та атмосферні локації, що підвищує стабільність геймплею та художній контроль.

Однак в контексті теми роботи, пов'язаної з процедурною генерацією ігрових карт, цей метод виділяється особливою актуальністю. Процедурна генерація дозволяє автоматизувати процес створення карт, забезпечуючи необмежену кількість унікальних ігрових досвідів для гравців. Це особливо корисно в іграх, де ключовим елементом є різноманітність і переіграбельність, таких як roguelike або відкритий світ. Такий підхід дозволяє розробникам створювати захоплюючі, неочікувані та непередбачувані ігрові світи, що залучає і утримує увагу гравців на тривалий час.

Отже, в контексті сучасної ігрової індустрії, процедурна генерація ігрових карт є актуальним та важливим інструментом для створення унікальних, захоплюючих та надзвичайно переіграбельних ігрових досвідів.

2 СКЛАДАЮЧА МЕТОДІВ ГЕНЕРАЦІЇ 2D КАРТ МІСЦЕВОСТІ

2.1 Опис методів генерації

Процедурна генерація карт (ПГК) - це метод створення карт для 2D ігор, який використовує алгоритми для автоматичного генерування унікальних ігрових світів. На відміну від статичних карт, що створюються вручну, ПГК дає можливість генерувати безліч карт, роблячи ігровий процес більш динамічним і повторюваним.

Це потрібно для того щоб:

– генерувати безліч унікальних карт, роблячи кожне проходження гри неповторним;

– значно економити час розробників, адже їм не потрібно вручну створювати кожну карту;

– збільшувати різноманіття карт, що робить ігровий процес більш цікавим і змушує гравців проходити гру знову і знову.

Процедурна генерація карт - це потужний інструмент, який може використовуватися для створення різноманітних і цікавих ігрових світів. ПГК може економити час розробників і значно підвищувати реіграбельність ігрового процесу.

2.2 Аналіз складових методів генерації

Цей розділ присвячений аналізу складових методів генерації карт. У ньому буде розглянуто принципи, що лежать в основі алгоритмів ПГК, а також їхні переваги та недоліки. Окрім цього буде представлений псведокод, що реалізує дані методи.

Цей розділ буде корисним для розробників ігор, які хочуть використовувати ПГК у своїх проектах, дослідників, які цікавляться

алгоритмами генерації карт, студентів, які вивчають комп'ютерні науки або математику.

Далі будуть розглянуті деякі методи генерації з їх реалізацією.

2.2.1 Шум Перліна (Perlin noise)

Шум Перліна - це алгоритм, що використовується для генерації плавних переходів між різними типами ландшафту. Він заснований на наступних математичних принципах:

Шум Перліна генерує градієнтне поле, яке описує напрямок та величину зміни шуму в кожній точці. Градієнтне поле генерується випадковим чином, але з певними обмеженнями, що забезпечують плавність переходів.

Для отримання значення шуму в конкретній точці використовується інтерполяція. Інтерполяція - це метод обчислення значення функції в точці, де значення функції невідоме, на основі значень функції в сусідніх точках.

Шум Перліна використовує декілька функцій шуму, які описують різні типи шуму, такі як гладкий шум, турбулентний шум, і т.д. Функції шуму - це математичні формули, які генерують випадкові числа з певними характеристиками.

Алгоритм шуму Перліна генерує шум покроково, додаючи шум з різних функцій шуму на кожному кроці. Алгоритм використовує інтерполяцію для обчислення значення шуму в кожній точці.

До переваг шуму Перліна належать наступні:

- плавність переходів;
- різноманіття;
- контрольованість.

До недоліків шуму Перліна належать наступні:

- складність реалізації;
- можливість генерування нереалістичних ландшафтів.

Градієнтне поле в шумі Перлина описує напрямок та величину зміни шуму в кожній точці. Це векторне поле, де кожен вектор вказує на напрямок найбільшого зростання шуму, а його довжина відповідає величині цього зростання.

Інтерполяція - це метод обчислення значення функції в точці, де значення функції невідоме, на основі значень функції в сусідніх точках. В шумі Перлина інтерполяція використовується для обчислення значення шуму в кожній точці на основі значень шуму в градієнтній сітці.

Шум Перлина використовує декілька функцій шуму, які описують різні типи шуму, такі як гладкий шум, турбулентний шум, і т.д. Функції шуму - це математичні формули, які генерують випадкові числа з певними характеристиками. Перевірятися буде саме гладкий шум. Він описується наступним чином:

Ось типова реалізація даного методу з використанням псевдокоду:

Спершу потрібно провести ініціалізацію. Приблизна реалізація зображена нижче.

```
for i in range(width):
 for j in range(height):
   grid[i][j] = 0
```

```
де:
gradient - масив, що описує градієнтне поле.
random_unit_vector - функція, що генерує випадковий одиничний вектор.
```

Код потрібен для того, щоб здійснити ініціалізацію масиву `grid` згідно з вказаним градієнтним полем та випадково згенерованими значеннями.

Далі створимо градієнтне поле. Реалізація зображена нижче.

```
for i in range(width):
  for j in range(height):
    gradient[i][j] = random_unit_vector()
```

```
де:
gradient - масив, що описує градієнтне поле.
random_unit_vector - функція, що генерує випадковий одиничний вектор.
```

Код потрібен для створення градієнтного поля, використовуючи функцію `random_unit_vector` для генерації випадкових одиничних векторів. Таким чином визначається функція random_unit_vector, яка генерує випадковий одиничний вектор. Встановлюються початкові значення для ширини та висоти масиву gradient. Заповнюється градієнтне поле gradient випадковими одиничними векторами, використовуючи функцію random_unit_vector. Виводиться отримане градієнтне поле для перевірки.

Опишемо функцію інтерполяції. Реалізація зображена нижче.

```
Func interpolate(grid, x, y):
  x0 = int(math.floor(x))
  y0 = int(math.floor(y))
  x1 = x0 + 1
  y1 = y0 + 1
  s00 = grid[x0][y0]
  s01 = grid[x0][y1]
  s10 = grid[x1][y0]
  s11 = grid[x1][y1]
  dx = x - x0
  dy = y - y0
  return (s00 * (1 - dx) * (1 - dy) +
          s01 * (1 - dx) * dy +
          s10 * dx * (1 - dy) +
          s11 * dx * dy)
```

де:
`dx` - зміщення по осі x.
`dy` - зміщення по осі y.

Ця функція інтерполяції отримує значення згенерованого масиву `grid` у певній точці `(x, y)` за допомогою білінійної інтерполяції.

Спочатку визначаються координати чотирьох найближчих точок у масиві grid для інтерполяції: (x0, y0), (x0, y1), (x1, y0), (x1, y1). Далі отримуються значення в цих точках: s00, s01, s10, s11 та обчислюється зміщення dx та dy між вхідною точкою (x, y) та найближчою точкою згідно до масиву grid. Використовуючи білінійну інтерполяцію, обчислює інтерпольоване значення вхідної точки (x, y) за допомогою значень у чотирьох найближчих точках та їх зміщень dx та dy. В кінці повертається інтерпольоване значення.

В результаті проведемо обчислення для кожної точки в сітці. Обчислення виглядають наступним чином:

```
noise = 0
for octave in range(octaves):
  frequency = 2^octave
  amplitude = persistence^octave
  noise += amplitude * interpolate(grid, x + frequency * dx, y + frequency * dy)
```

де:
noise - значення шуму в даній точці.
octaves - кількість октав.
persistence - персистентність.
interpolate - функція інтерполяції.
dx - зміщення по осі x.
dy - зміщення по осі y.

Цей код виконує обчислення шуму для кожної точки в сітці шляхом сумування шумів різних октав, а саме ініціалізується змінна noise для збереження загального шуму в даній точці. Проходиться по кількості октав, використовуючи змінну octaves. Обчислюється частота (frequency) та амплітуда (amplitude) для кожної октави. Обчислюється зміщення dx та dy для кожної октави. Використовуючи функцію інтерполяції interpolate, обчислюється значення шуму для кожної октави та додає його до загального шуму. Повертається загальне значення шуму в даній точці.

2.2.2 Направлений тунель (Directional tunnel)

Метод направлений тунель (Directional tunnel) використовується для генерації 2D карт з тунелями, що йдуть у певному напрямку. Цей метод використовує наступні математичні принципи:

– створюється квадратна сітка з фіксованим розміром комірок;

– кожній комірці сітки присвоюється значення, що описує тип ґрунту (наприклад, камінь, земля, вода);

– вибирається випадковий напрямок тунелю;

– цей напрямок задається вектором з двома координатами (x, y);

– починаючи з вільної комірки, алгоритм прокладає тунель у заданому напрямку;

– на кожному кроці алгоритм перевіряє, чи вільна сусідня комірка в заданому напрямку;

– якщо сусідня комірка вільна, тунель продовжується в цьому напрямку;

– якщо сусідня комірка зайнята, тунель робить поворот на 90 градусів у випадковому напрямку;

– алгоритм продовжує прокладати тунель, поки не досягне краю сітки або не зустріне непереборну перешкоду (наприклад, твердий камінь).

Після прокладання основного тунелю до нього можна додати різні деталі, такі як бічні тунелі, кімнати, печери тощо.

Ці деталі можна генерувати випадковим чином або за допомогою спеціальних алгоритмів.

Ось типова реалізація даного методу з викоритсанням псевдокоду:

Спершу генерується сітка. Код зображений нижче:

```
for i in range(width):
  for j in range(height):
    grid[i][j] = generate_terrain_type(i, j)
```

```
де:
grid - масив, що описує сітку.
width - ширина сітки.
height - висота сітки.
generate_terrain_type - функція, що генерує тип ґрунту для комірки.
```

Цей код дозволяє створити сітку із зазначеними розмірами (`width` та `height`) та заповнити її типами ґрунту за допомогою функції `generate_terrain_type`.

Далі сам метод. Код зображений нижче:

```
while not reached_edge and not hit_obstacle:
  next_cell = grid[position.x + direction.x][position.y + direction.y]
  if next_cell == FREE:
    position += direction
  else:
    direction = random_rotation(direction)
```

```
де:
position - поточне положення тунелю.
next_cell - тип ґрунту в наступній комірці.
FREE - значення, що описує вільну комірку.
hit_obstacle - прапорець, що
```

Цей код представляє метод для переміщення тунелю в сітці, доки не буде досягнуто краю сітки або не буде зіткнутося з перешкодою.

В цілому, метод направлений тунель можна зрозуміти та реалізувати на базовому рівні без знання складних математичних концепцій. Проте, для його поглибленого розуміння та реалізації складніших варіантів знадобиться знання алгоритмів, векторів, та структур даних. Він не потребує глибоких знань з математики, але дає можливість зрозуміти основні принципи генерації карт.

2.2.3 Клітинний автомат (Cellular Automata)

Метод клітинного автомата є потужним інструментом для генерації карт 2D. Цей метод полягає в тому, щоб моделювати складний поведінковий візуальний вигляд за допомогою простих правил на основі стану окремих "клітин" або елементів.

Основна ідея полягає в наступному:

Карта починається з певного початкового стану, який може бути випадковим або заданим вручну, тобто **генерується початковий стан**.

Визначається набір правил, які визначають, як буде змінюватися стан кожної клітини в залежності від її поточного стану та стану сусідніх клітин.

Застосовуючи визначені правила, оновлюється стан кожної клітини на основі її поточного стану та стану сусідніх клітин.

Процес оновлення стану клітин повторюється протягом декількох ітерацій або до досягнення задовільного візуального результату.

Останній крок - візуалізація отриманої карти.

Ось типова реалізація даного методу з викоритсанням псевдокоду:

Спершу буде ініційовано поточний стан сітки та описано правила згідно з якими буде змінюватись стан. Код зображений нижче:

```
grid = generate_initial_state(width, height)

def rules(current_state, north, south, east, west):
 #Визначаємо правила
  return newstate

   де:
current_state - Поточний стан комірки.
north - Стан північної сусідньої комірки.
south - Стан південної сусідньої комірки.
east - Стан східної сусідньої комірки.
west - Стан західної сусідньої комірки.
```

Цей код визначає ініціальний стан сітки та правила, згідно з якими буде змінюватися стан кожної комірки в майбутньому. Змінна grid ініціалізується за допомогою функції generate_initial_state, яка генерує початковий стан сітки за вказаними розмірами. Функція rules приймає поточний стан комірки та стани її сусідніх комірок на північ, південь, схід та захід. В тілі функції визначаються правила, за якими буде змінюватися стан комірки. У цьому коді наведено простий приклад правил: якщо північна сусідня комірка жива, а поточна мертва, то стан поточної комірки змінюється на живий, і навпаки.

Після цього проводиться необхідна кількість ітерацій оновлення та відображається результат. Код зображений нижче:

```
for _ in range(iterations):
  for i in range(width):
    for j in range(height):
      #Оновлення стану
      new_state = rules[grid[i][j]][get_neighbors(grid, i, j)]
      grid[i][j] = new_state

visualize_grid(grid)
```

Цей код виконує необхідну кількість ітерацій оновлення стану сітки за заданими правилами та відображає результат. Зовнішній цикл for здійснює задану кількість ітерацій оновлення сітки (зазначену змінною iterations). Внутрішні цикли for проходяться по кожній комірці сітки за допомогою двох

змінних i та j. Для кожної комірки виконується оновлення стану за допомогою правил, визначених функцією rules. Новий стан комірки зберігається у відповідній комірці сітки. Після завершення всіх ітерацій відображається результат оновлення сітки за допомогою функції visualize_grid.

Околиця Мура (Moore Neighbourhood) є одним із способів визначення сусідства клітин у клітинних автоматах.

В Околиці Мура кожна клітина має вісім сусідів: по чотири по діагоналі та чотири по вертикалі/горизонталі. Це означає, що клітина знаходиться в центрі квадрата 3x3, де вона сама є центром, і з'єднана з усіма восьми клітинами навколо неї.

Використання Околиці Мура в клітинних автоматах для зглажування початкової генерації клітин дозволяє визначити, які клітини впливають на зміну стану даної клітини.

Нижче подано псевдокод реалізації методу Околиці Мура для визначення сусідів клітини у клітинному автоматі:

```
function getMooreNeighborhood(cell_x, cell_y, grid):
    neighbors = []
    for dx from -1 to 1:
        for dy from -1 to 1:
            if dx == 0 and dy == 0:
                continue  // Skip the cell itself
            neighbor_x = cell_x + dx
            neighbor_y = cell_y + dy

            if neighbor_x >= 0 and neighbor_x < grid.width and neighbor_y >= 0 and
neighbor_y < grid.height:
                neighbors.append(grid[neighbor_x][neighbor_y])
    return neighbors
```

де:
cell_x, cell_y - Координати поточної комірки, для якої ми шукаємо сусідів.
grid - Сітка клітин, у якій розташована поточна комірка та її сусіди.
neighbors - Список, в який зберігаються сусіди поточної комірки за околицею Мура.
dx, dy - Змінні, які використовуються для ітерації через можливі зміни в координатах по горизонталі та вертикалі навколо поточної комірки.
neighbor_x, neighbor_y - Координати потенційного сусіда, обчислені шляхом додавання dx та dy до координат поточної комірки.
grid.width, grid.height - Ширина та висота сітки клітин, що використовуються для перевірки того, чи знаходяться потенційні сусіди в межах сітки.

Цей код створює функцію `get_moore_neighborhood`, яка приймає координати поточної комірки (`cell_x`, `cell_y`) та сітку `grid`. Вона повертає список сусідів поточної комірки за методом Околиці Мура.

Околиця фон Неймана (von Neumann Neighbourhood) в контексті зглажування початкової генерації клітинних автоматів може використовуватися для визначення сусідів кожної клітини у сітці клітин.

Коли ми говоримо про зглажування початкової генерації клітинних автоматів, ми маємо на увазі процес встановлення початкових станів клітин у сітці перед початком самого процесу клітинного автомата. Використання околиці фон Неймана в цьому контексті означає, що ми розглядаємо тільки чотири основних напрямки сусідства для кожної клітини.

Це може бути корисно для створення більш простих та менш складних систем клітинних автоматів, де потрібно враховувати лише прямий вплив сусідів, розташованих по вертикалі або горизонталі, а інші напрямки вважаються непотрібними або небажаними.

Отже, використання околиці фон Неймана може спростити обчислення та зменшити кількість взаємодій між клітинами, що може бути корисно в деяких конкретних випадках або для досягнення певної початкової конфігурації у системі клітинних автоматів. Реалізація методу зображена нижче:

```
function getVonNeumannNeighborhood(cell_x, cell_y, grid):
    neighbors = []
    if cell_y - 1 >= 0:
        neighbors.append(grid[cell_x][cell_y - 1])
    if cell_y + 1 < grid.height:
        neighbors.append(grid[cell_x][cell_y + 1])
    if cell_x - 1 >= 0:
        neighbors.append(grid[cell_x - 1][cell_y])
    if cell_x + 1 < grid.width:
        neighbors.append(grid[cell_x + 1][cell_y])
    return neighbors
```

де:

`cell_x, cell_y` - Координати поточної комірки, для якої ми шукаємо сусідів.
`grid` - Сітка клітин, у якій розташована поточна комірка та її сусіди.
`neighbors` - Список, в який зберігаються сусіди поточної комірки за околицею фон Неймана.
`grid.width, grid.height` - Ширина та висота сітки клітин, що використовуються для перевірки того, чи знаходяться потенційні сусіди в межах сітки.

Цей псевдокод використовує метод Околиці фон Неймана для визначення сусідів клітини в клітинному автоматі. Він розглядає лише сусідні клітини, які знаходяться вгорі, внизу, ліворуч та праворуч від поточної клітини. Цей код створює функцію `get_von_neumann_neighborhood`, яка приймає координати поточної комірки (`cell_x`, `cell_y`) та сітку `grid`. Вона повертає список сусідів поточної комірки за методом Околиці фон Неймана.

2.2.4 Випадкове блукання (Random Walk)

Випадкове блукання (Random Walk) - це процес, у якому об'єкт рухається крок за кроком у випадковому напрямку.

У контексті генерації 2D карт, випадкове блукання може бути використане для створення складних і непередбачуваних ландшафтів або теренів. Процес може починатися з випадково обраної початкової позиції, а потім об'єкт (який може представляти рельєф терену) рухатиметься випадковим чином, кожен раз змінюючи своє положення в залежності від випадкового напрямку.

Цей метод може бути використаний для створення непередбачуваних та унікальних ландшафтів, які можуть бути використані у відеоіграх, симуляціях або візуалізаціях. Він дає можливість створити реалістичні та відтворювані результати, які можуть відповідати різним умовам та потребам.

```
function randomWalk(steps):
    x = 0
    y = 0
    path = []
    path.append((x, y))
    for i from 1 to steps:
        direction = randomDirection()
        if direction == "up":
            y = y + 1
        else if direction == "down":
            y = y - 1
        else if direction == "left":
            x = x - 1
        else if direction == "right":
```

```
            x = x + 1
        path.append((x, y))
    return path

function randomDirection():
    randomNumber = random(1, 4)
    if randomNumber == 1:
        return "up"
    else if randomNumber == 2:
        return "down"
    else if randomNumber == 3:
        return "left"
    else:
        return "right"

      де:
steps - Кількість кроків, яку повинен зробити об'єкт під час блукання.
x, y - Поточні координати об'єкта на площині.
path - Список координат, які представляють шлях об'єкта під час блукання.
direction - Змінна, яка зберігає напрямок наступного кроку випадкового блукання.
```

Цей код визначає дві функції: `random_walk`, яка здійснює блукання об'єкта на площині, та `random_direction`, яка вибирає випадковий напрямок для кожного кроку блукання. В результаті виконання функції `random_walk` повертається список координат, які представляють шлях об'єкта під час блукання. Цей метод може бути використаний для створення реалістичних та унікальних ландшафтів у візуалізаціях або ігрових середовищах.

3 ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ

3.1 Методологія проведення експерименту

Методологія у роботі зосереджувалась на тому, як саме виконувались дослідження та аналіз алгоритмів процедурної генерації карт у 2D іграх. Ось загальний опис методології:

– спочатку було здійснено огляд літератури та існуючих алгоритмів процедурної генерації карт у 2D іграх. На основі цього огляду було обрано конкретні алгоритми для подальшого аналізу;

– на основі обраних алгоритмів розроблювались прототипи програмного забезпечення для генерації 2D карт. Кожен прототип використовував один з обраних алгоритмів;

– далі було визначено параметри кожного алгоритму, які можна змінювати для отримання різних типів карт. Це може включати параметри, такі як розмір карти, кількість об'єктів на карті, ступінь деталізації тощо;

– створено тестовий набір, що складається з різних типів карт, згенерованих кожним з розглянутих алгоритмів. Цей набір буде використовуватися для оцінки якості та ефективності кожного алгоритму;

– проведено експериментальне порівняння різних алгоритмів за допомогою тестового набору. Оцінка результатів буде здійснюватися за допомогою кількісних метрик, таких як швидкість генерації, рівень деталізації, рівень складності тощо;

– аналіз отриманих результатів експерименту та формулювання висновків щодо ефективності та придатності кожного з обраних алгоритмів для процедурної генерації карт у 2D іграх.

Ця методологія дозволить систематично дослідити різні алгоритми процедурної генерації карт у 2D іграх і зробити обґрунтований вибір найбільш ефективного та придатного для вашої задачі.

3.2 Обгрунтування середовища розробки

В роботі є ключовим етапом вибір конкретної платформи для розробки додатку. Оскільки ваша тема роботи полягає у дослідженні та аналізі алгоритмів процедурної генерації карт у 2D іграх, обрана платформа для розробки має бути обґрунтована з урахуванням специфіки задачі.

Unity вибирається через наступні причини:

– Unity - одна з найбільш популярних та потужних платформ для розробки ігор, яка забезпечує широкий набір інструментів і можливостей для створення якісних ігрових додатків;

– хоча Unity спочатку була більш відома як ігровий движок для розробки 3D ігор, вона також надає потужні засоби для розробки 2D ігор. Зокрема, Unity має вбудовані засоби для роботи з 2D графікою, фізикою та анімацією, що дозволить легко реалізувати ваші цілі;

– Unity має велику та активну спільноту розробників і дуже добру документацію. Це означає, що ви зможете легко знайти відповіді на свої запитання, а також скористатися готовими рішеннями і підказками;

– Unity дозволяє розробляти ігри для різних платформ, включаючи ПК, мобільні пристрої та ігрові консолі. Це робить його ідеальним вибором для створення ігор, які можуть бути випущені на різних пристроях;

– Unity має велику кількість розширень і активно розвивається. Це дає вам можливість легко і швидко додавати нові функції до вашого проекту за допомогою сторонніх інструментів та плагінів.

Отже, обгрунтування вибору Unity для розробки вашої магістерської роботи зосереджується на його потужних можливостях, підтримці 2D розробки, широкому спектрі ресурсів для навчання та розвитку, а також кросплатформенній природі, яка дозволить вам працювати з різними типами пристроїв і платформ.

3.3 Специфікація програмного забезпечення

Специфікація програмного забезпечення - це документ, який визначає вимоги до функціональності, архітектури та інших аспектів програмного продукту. Вона слугує основою для розробки, тестування та оцінки програмного забезпечення. У межах даної теми специфікація визначає параметри і функціональні можливості програми, яка досліджує та аналізує алгоритми генерації карт у 2D іграх, зокрема, її засоби реалізації, вимоги до інтерфейсу користувача, мови програмування тощо.

Таким чином вона має наступний вигляд:

– в основі реалізації програмного забезпечення є мова програмування C#, оскільки Unity використовує цю мову для розробки ігрових додатків;

– у реалізації програмного забезпечення використані інструменти, надані Unity, такі як Scene Editor для розміщення об'єктів на сцені, Inspector для налаштування параметрів об'єктів, та інші;

– програмне забезпечення взаємодіє з різними компонентами Unity, такими як Rigidbody для реалізації фізики, Colliders для обробки зіткнень, та інші, для створення та обробки генерованих карт;

– реалізація програмного забезпечення базуватися на різних алгоритмах процедурної генерації карт, які будуть імплементовані за допомогою скриптів C# в середовищі Unity;

– для взаємодії з програмним забезпеченням можна створити інтерфейс користувача, можливо, з використанням інструментів Unity, таких як UI Toolkit, для представлення налаштувань та результатів генерації карт.

Ця специфікація визначає основні аспекти програмного забезпечення, які будуть використані для дослідження та аналізу алгоритмів процедурної генерації карт у 2D іграх з використанням Unity.

3.4 Реалізація методів генерації

Як вже було зазначено раніше для демонстрації було використано Unity. Для генерації було створено скрипт, в якому обирається сам метод генерації та налаштовуються його параметри такі як розмір карти, типу ширина і довжина, тайли для землі та води, щоб було зручного переглядати результат, та параметри, що необхідні для налашутвання методів генерації. Інспектор зі скриптом зображений нижче на рис. 3.1:

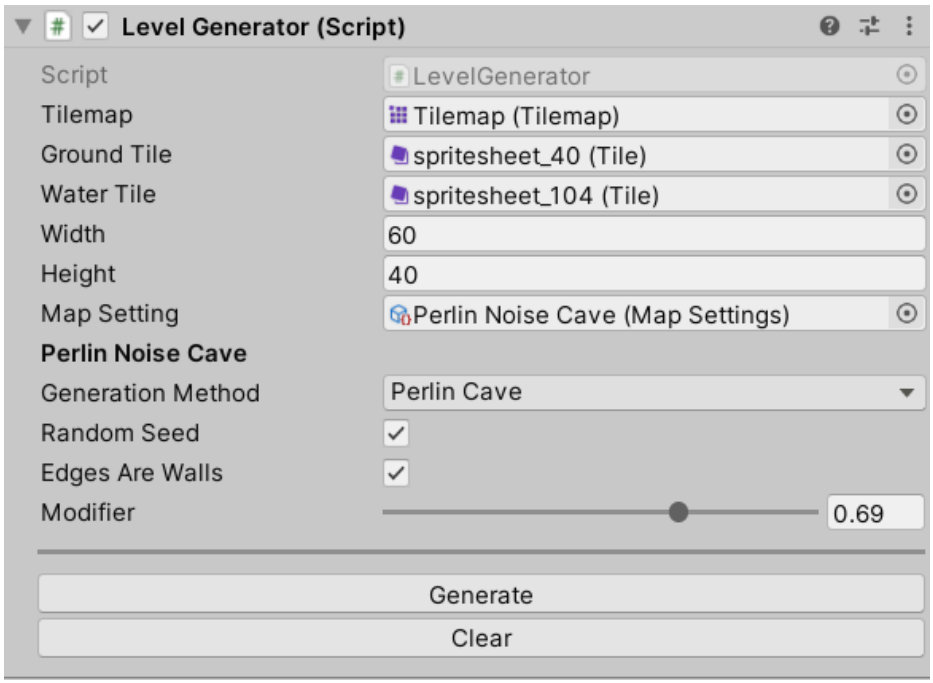

Рисунок 3.1 – Інспктор з параметрами генерації

На рисунку зображено інспектор Unity, де налаштовуються параметри для генерації карти методом. Нижче наведено опис кожного параметра:

– Script: вказує на скрипт, прикріплений до об'єкта, який відповідає за генерацію рівня. У цьому випадку це LevelGenerator;

– Tilemap: вказує на об'єкт типу Tilemap, який визначає, де будуть розміщені плитки на карті;

– Ground Tile: задає плитку, яка буде використовуватися для землі;

– Water Tile: задає плитку, яка буде використовуватися для води;

– Width: визначає ширину карти;

– Height: визначає висоту карти;

– Map Setting: вказує на налаштування карти, що використовуються для генерації;

– Generation Method: Вибір методу генерації. У цьому випадку вибрано Perlin Noise;

– Random Seed: Прапорець для використання випадкового зерна. Якщо встановлено, використовується випадкове число для генерації карти;

– Edges Are Walls: Прапорець, що визначає, чи будуть краї карти стінами;

– Modifier: Повзунок для налаштування модифікатора, що впливає на генерацію карти;

– Generate: Кнопка для запуску процесу генерації карти;

– Clear: Кнопка для очищення згенерованої карти.

Ці параметри дозволяють налаштувати генерацію карти різними методами та забезпечити контроль над зовнішнім виглядом та структурою згенерованого рівня.

3.4.1 Реалізація алгоритму шум перліна (Perlin noise)

Перліновий шум характеризується плавними змінами значень, що робить його ідеальним для моделювання нерівностей рельєфу, хмар, води та інших природних структур. Різні октави (покоління) Перлінового шуму дозволяють регулювати рівень деталізації і складності згенерованих карт, створюючи від найгладкіших до найбільш детальних текстур. Приклад використання ццього методу зображений нижче на рис. 3.2.

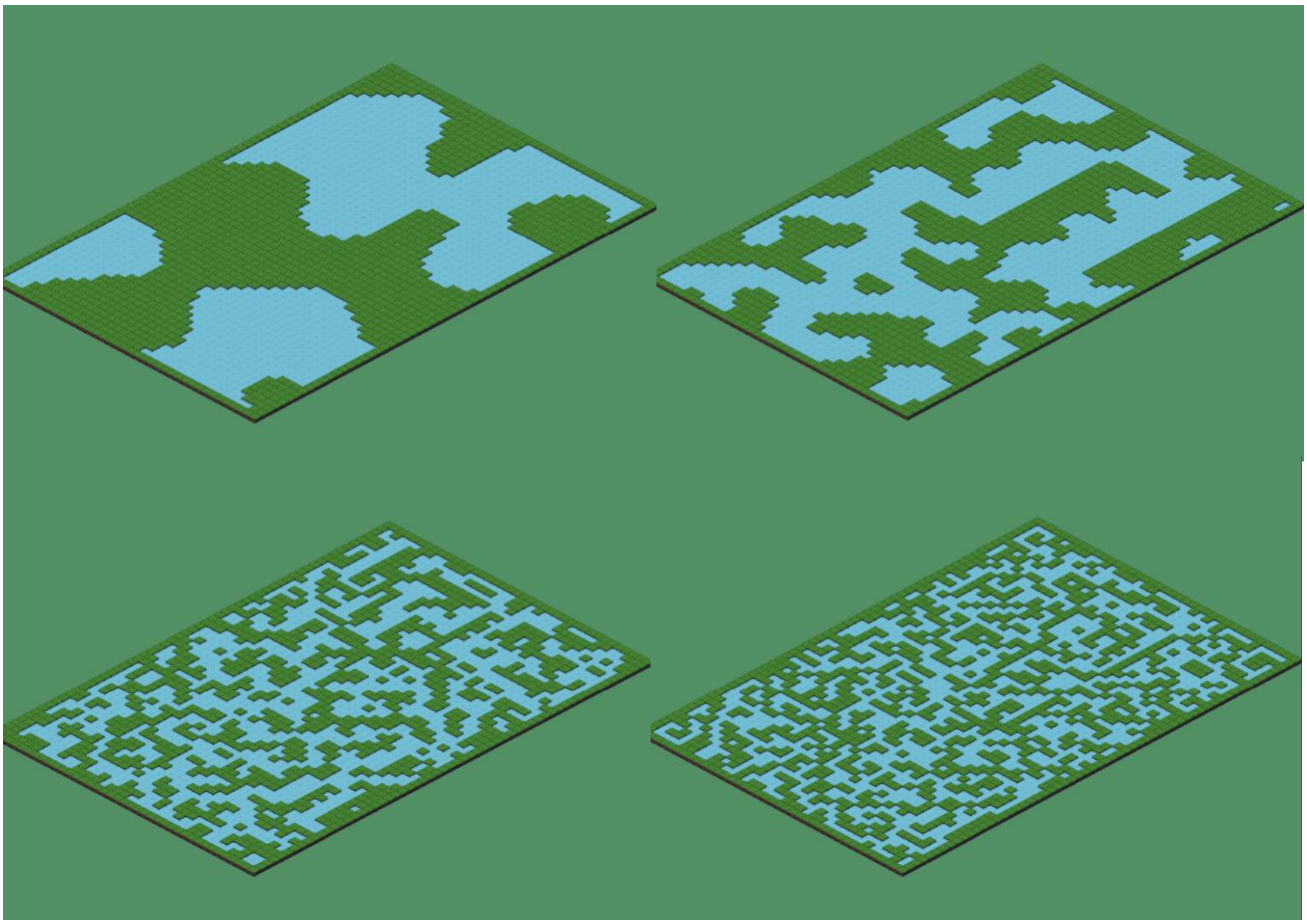

Рисунок 3.2 – Декілька варіантів генерації методом Perlin noise

На рисунку зображено чотири різні генерації карти за допомогою методу Perlin Noise з використанням різних октав. Кожна карта демонструє різну текстуру і деталізацію, яка досягається за рахунок зміни кількості октав в алгоритмі Perlin Noise. Ось детальний опис кожної з них:

Верхній лівий кут:

– ця карта має найменшу деталізацію і великі, плавні переходи між землею (зелені області) і водою (сині області);

– використовується менша кількість октав, що створює великі, менш деталізовані області;

– вода та земля утворюють великі, неоднорідні контури.

Верхній правий кут:

– ця карта має трохи більше деталізації в порівнянні з першою;

– використовується середня кількість октав, що створює більш розбиті та детальні області води і землі;

– переходи між землею і водою більш різкі, але все ще залишаються відносно плавними.

Нижній лівий кут:

– ця карта має ще вищу деталізацію;

– використовується більша кількість октав, що створює дрібніші та більш деталізовані структури;

– вода і земля утворюють мозаїкоподібні візерунки з дрібними фрагментами.

Нижній правий кут:

– це карта з найвищою деталізацією серед усіх чотирьох;

– використовується максимальна кількість октав, що створює високодеталізовані та складні візерунки;

– вода і земля перемішані в дрібні, майже піксельні структури з дуже різкими переходами між ними.

Загальні спостереження:

– від верхнього лівого кута до нижнього правого кута зростає деталізація і складність візерунків;

– більша кількість октав призводить до збільшення деталізації та складності текстур;

– менша кількість октав створює більш плавні та великі переходи, тоді як більша кількість октав створює різкіші та детальніші переходи.

Ці різні генерації можуть бути використані для створення різних типів ігрових ландшафтів залежно від потреб розробника.

3.4.2 Реалізація алгоритму направлений тунель (Directional tunnel)

Як вже вказувалось вище метод Directional Tunnel є одним із способів генерації двовимірних карт, який використовується для створення лабіринтів або тунелів у процедурно згенерованих світах. Цей підхід базується на визначенні напрямків руху для побудови зв'язних шляхів і структур, що створює реалістичні та складні маршрути. Directional Tunnel дозволяє налаштовувати різні параметри, такі як кривизна, розгалуження та довжина тунелів, що сприяє створенню унікальних ігрових ландшафтів і середовищ. Це робить його популярним у розробці ігор та симуляцій, де важливі динамічні та цікаві карти.

Інспектор з параметрами для генерації даним методом зображено нижче на рис 3.3.

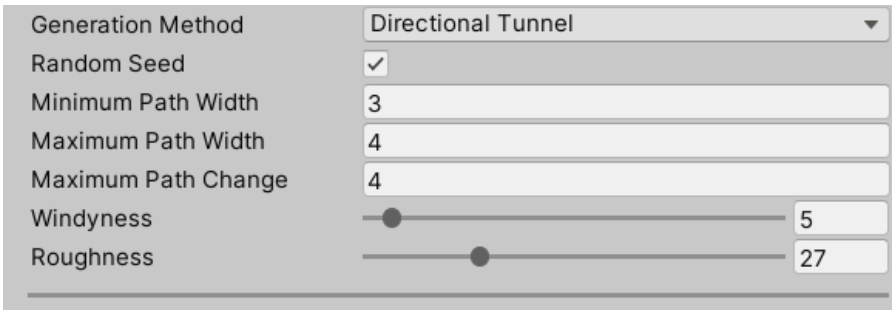

Рисунок 3.3 – Інспектор з параметрами для генерації за допомогою методу Directional tunnel

На другому скріншоті зображені параметри, які використовуються для генерації карти методом Directional Tunnel:

– Minimum Path Width: мінімальна ширина шляху, який буде створено;

– Maximum Path Width: максимальна ширина шляху, який буде створено;

– Maximum Path Change: максимальна кількість змін шляху, що визначає, наскільки часто шлях може змінювати напрямок;

– Windyness: параметр, що визначає "звивистість" шляху, тобто як часто і наскільки сильно шлях відхиляється від свого основного напрямку;

– Roughness: параметр, що визначає "шорсткість" шляху, тобто наскільки нерівним буде його контур.

Результати генерації карт зображені нижче на рис. 3.4.

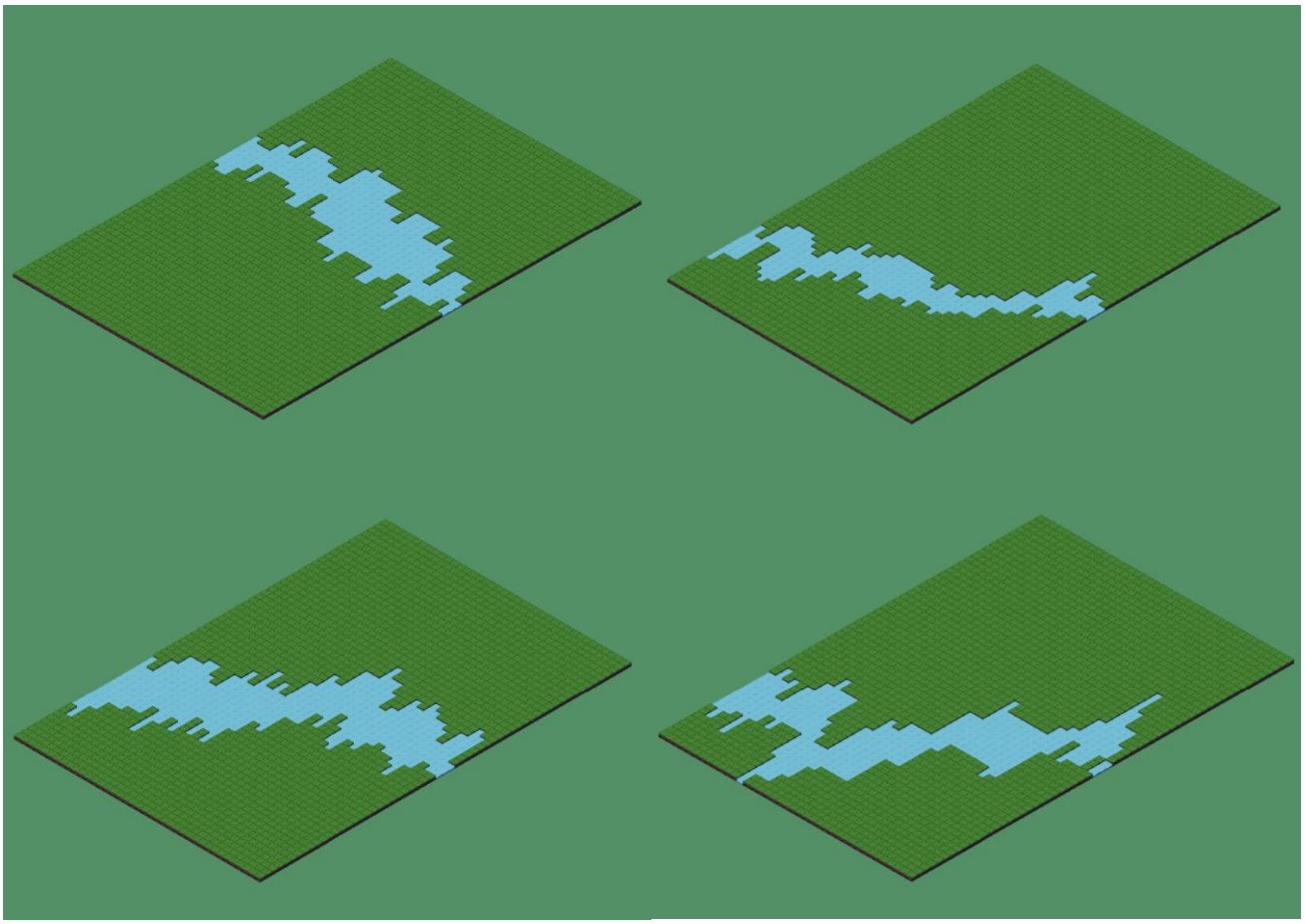

Рисунок 3.4 – Декілька варіантів генерації методом Directional tunnel

На рисунку зображено чотири різні генерації карти, створені методом Directional Tunnel з використанням різних комбінацій параметрів. Кожна карта відрізняється формою та структурою тунелю, що демонструє вплив зміни параметрів на кінцевий результат. Ось детальний опис кожної з них:

Верхній лівий кут:

– тунель має середню ширину та помірну звивистість;

– параметри можуть бути налаштовані так, що максимальна ширина шляху не сильно перевищує мінімальну, а зміни у напрямку тунелю не надто різкі;

– тунель виглядає досить прямолінійним з кількома помірними вигинами.

Верхній правий кут:

– тунель має трохи меншу ширину та більш звивистий характер;

– параметр Windyness збільшений у порівнянні з першою генерацією, що робить шлях більш звивистим;

– тунель має нерівномірний контур з частими змінами напрямку.

Нижній лівий кут:

– ця генерація має ширший тунель з менш звивистим шляхом;

– мінімальна та максимальна ширина шляху можуть бути збільшені, що забезпечує ширший тунель;

– тунель виглядає більш прямолінійним з незначними вигинами, можливо, параметр Windyness зменшений.

Нижній правий кут:

– тунель має найбільшу ширину та найвищу звивистість;

– максимальна ширина шляху значно перевищує мінімальну, а параметр Windyness встановлений на високий рівень;

– тунель має дуже нерівний контур з частими і значними змінами напрямку.

Ці різні генерації можуть бути використані для створення різних типів підземних лабіринтів або коридорів у грі, залежно від потреб розробника.

3.4.3 Реалізація алгоритму клітинний автомат (Cellular Automata)

В цому підрозділі будуть розглянуті два методи згладжування: Moore Neighbourhood та von Neumann Neighbourhood. Спочатку буде згенеровано карту до згладжування, а потім буде розглянуто кожен метод згладжування окремо. Для порівняння будуть використовуватись наступні критерії: структура детальність органічність функціональність. Але спершу розглянемо параметри що будуть використовуватись для алгоритмів зглажування. Інспектор з параметрами представлений на рис. 3.5.

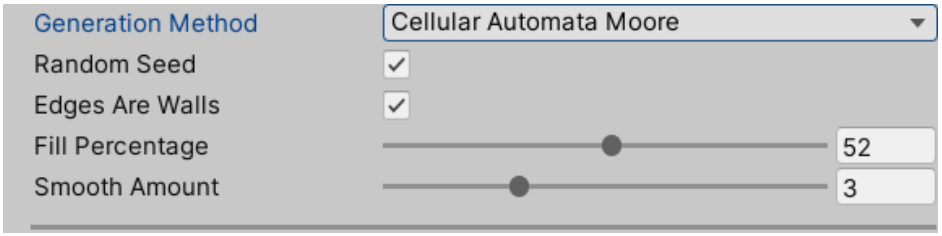

Рисунок 3.5 – Інспектор з параметрами для зглажування згенерованої карти за допомогою методів von Neumann Neighbourhood та Moore Neighbourhood

На даному рисунку продемонстровані параметри, які використовуються для генерації та згладжування карт.

Опис параметрів рисунку:

– Random Seed: прапорець, що визначає, чи використовувати випадкове зерно для генерації карти. Якщо встановлений, то кожен запуск генерації буде давати різні результати;

– Edges Are Walls: прапорець, що вказує, чи повинні краї карти бути стінами. Якщо встановлений, тоді всі краї карти будуть заповнені стінами;

– Fill Percentage: повзунок, що визначає відсоток заповнення карти стінами. Наприклад, значення 52 означає, що 52% карти буде заповнено стінами;

– Smooth Amount: кількість ітерацій згладжування, що визначає, скільки разів буде застосовано алгоритм згладжування.

Далі зображено на рисунку 3.6 зображено варіант згенерованої карти до зглажування.

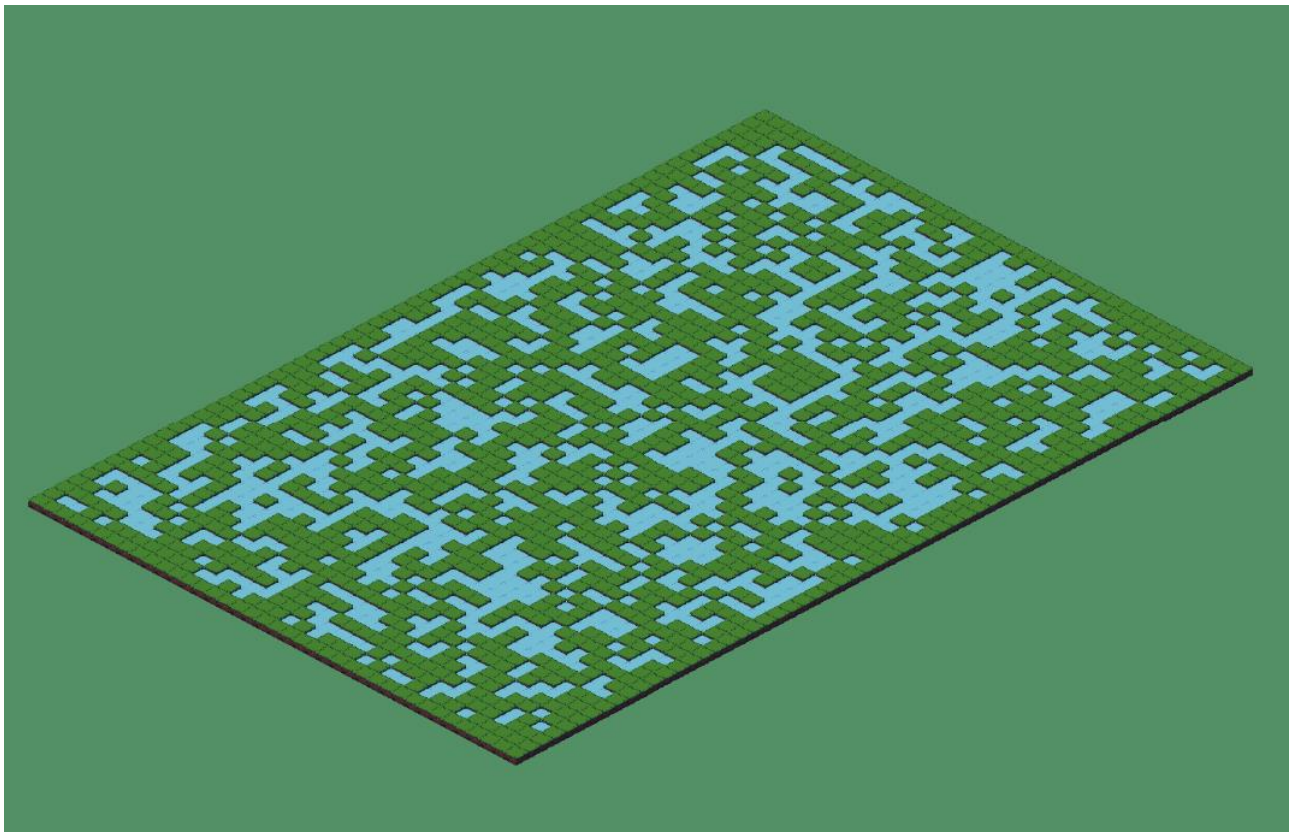

Рисунок 3.6 – Випадково згенерована карта

Для даної генераії було проведено оцінку вище вказаних параметрів:

– структура: карта демонструє випадковий і хаотичний розподіл зелених (суходіл) та блакитних (вода) клітинок.

– деталізація: велика кількість дрібних деталей та різких переходів між суходолом та водою.

– органічність: конфігурація клітинок виглядає штучно та неприродно через нерівномірні й випадкові форми областей.

– функціональність: така карта може бути корисною для створення складних лабіринтів або випадкових середовищ, де потрібна велика кількість дрібних деталей.

Реалізація алгоритму Околиця Мура (Moore Neighbourhood) як методу зглажування початкової генерації клітинних автоматів має вигляд зображений на рисунку 3.7.

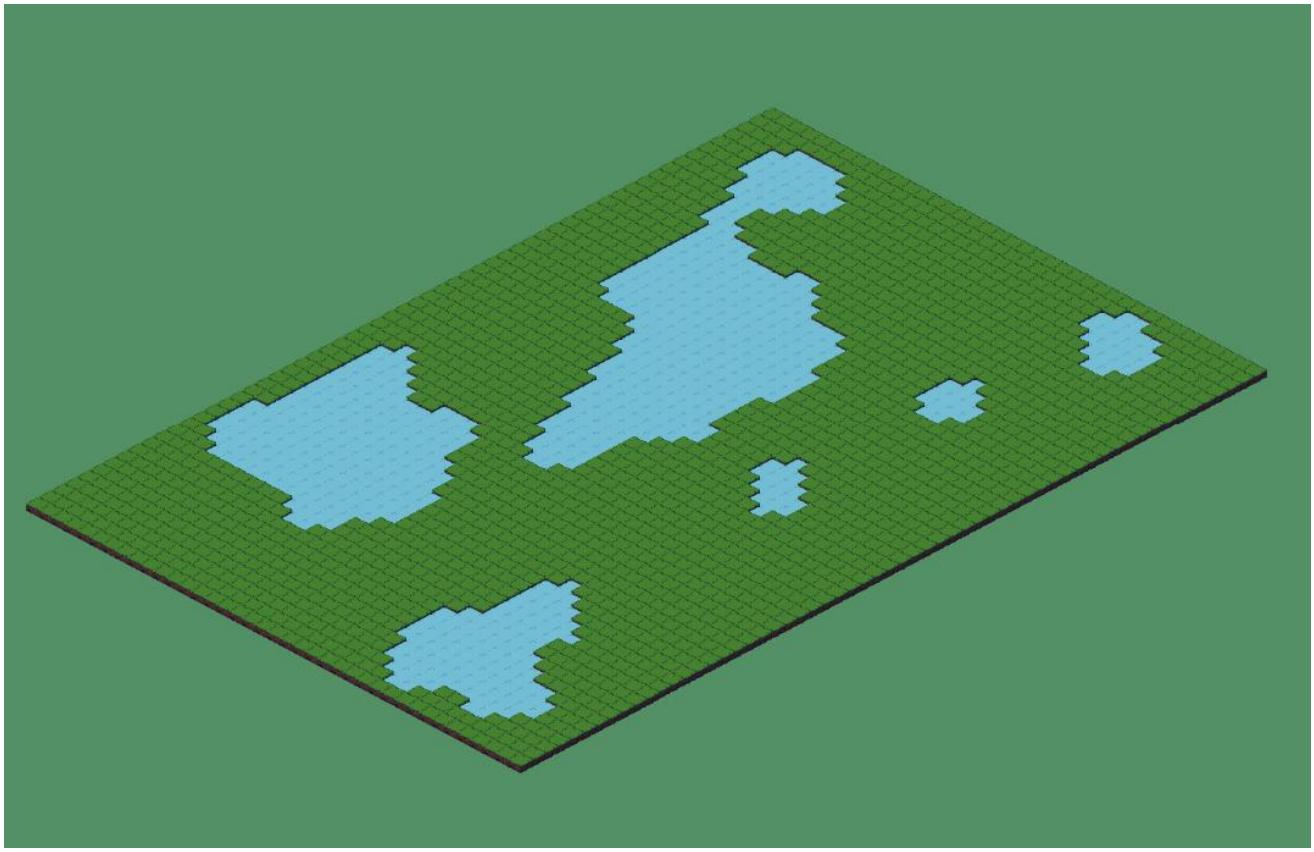

Рисунок 3.7 – Випадково згенерована карта після зглажування методом Moore Neighbourhood

Для даної генераії було проведено оцінку вище вказаних параметрів:

– структура: після застосування методу Moore Neighbourhood, карта стала значно гладкішою, з більшими, більш зв'язними областями води та суходолу;

– якісні зміни: згладжування зменшило кількість дрібних та розрізнених областей, об'єднавши їх у більші та більш чіткі зони;

– органічність: карта виглядає більш природно та реалістично, з плавними переходами між водою та суходолом, що нагадує природні ландшафти;

– функціональність: така карта підходить для створення відкритих просторів або природних середовищ, де потрібні плавні переходи та великі області.

Реалізація алгоритму околиця фон Неймана (von Neumann Neighbourhood) як методу зглажування початкової генерації клітинних автоматів має вигляд зображений на рисунку 3.8.

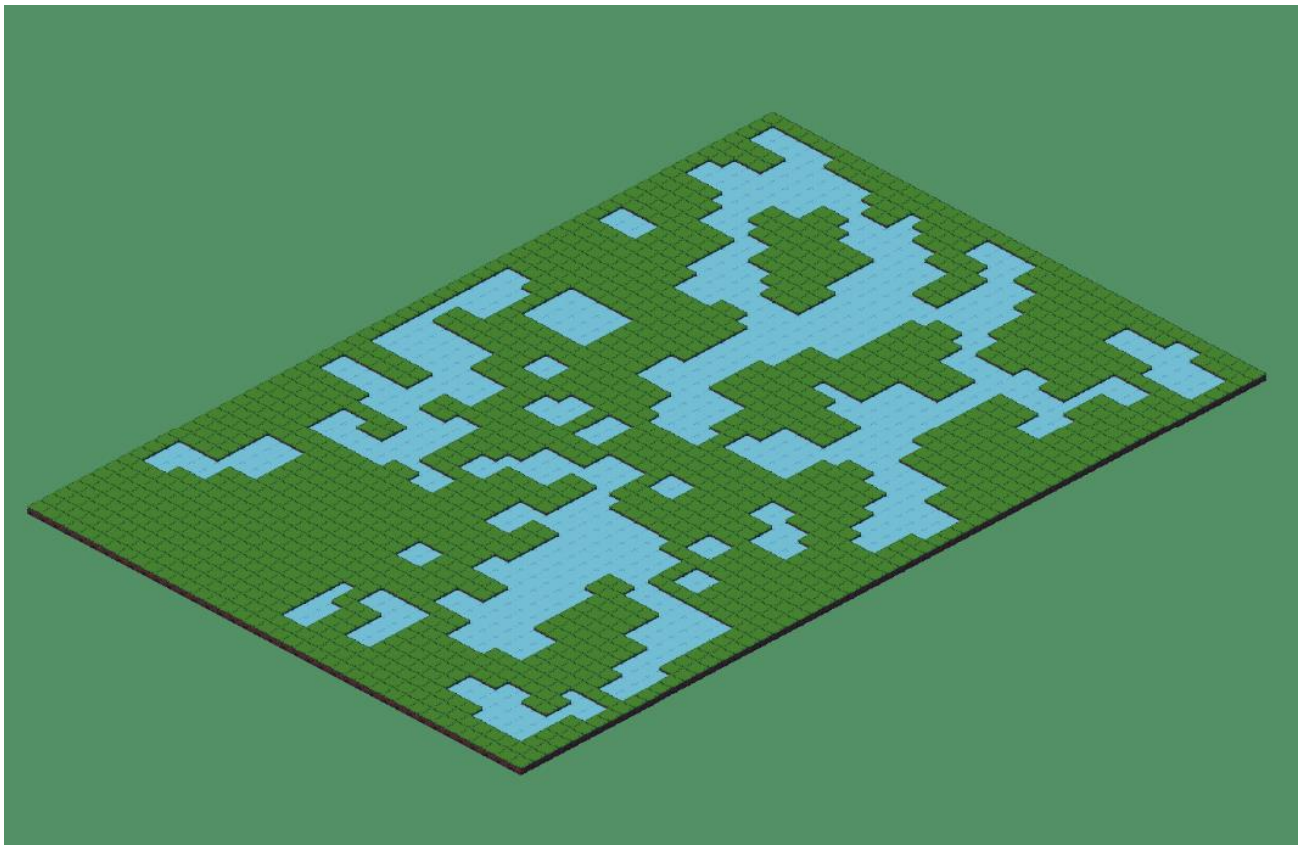

Рисунок 3.8 – Випадково згенерована карта після зглажування методом Von Neumann Neighbourhood

Для даної генераії було проведено оцінку вище вказаних параметрів:

– структура: після застосування методу Von Neumann Neighbourhood, карта стала більш структурованою, з більшими та більш зв'язними областями води та суходолу;

– якісні зміни: згладжування зменшило кількість дрібних та розрізнених областей, об'єднавши їх у більші та більш чіткі зони;

– органічність: карта виглядає більш природно, однак все ще має деякі різкі переходи, що робить її менш гладкою, ніж при застосуванні інших методів згладжування;

– функціональність: така карта підходить для створення середовищ зі складними структурами, де потрібні як великі, так і малі області.

Оскільки детальне порівняння всіх методів буде представлено нижче то тут буде порівняно лише основніхарактеристики методів зглажувсання.

Moore Neighbourhood: дає більш гладкі результати завдяки врахуванню більшої кількості сусідів, що робить перехід між стінами та порожніми областями більш поступовим.

von Neumann Neighbourhood: зберігає дещо більше структурності та різкості через врахування меншої кількості сусідів, що може бути корисним для створення більш чітких контурів у генерації карт.

Ці методи згладжування можуть бути використані для різних цілей у розробці ігор або інших застосунків, де потрібна генерація карт з різними рівнями деталізації та гладкості.

3.4.4 Реалізація алгоритму випадкового блукання (Random Walk)

У цьому розділі було розглянуто метод генерації карти під назвою Random Walk. Як вже вказувалось раніше, цей підхід використовується для створення процедурно згенерованих карт, таких як печери або лабіринти, за допомогою випадкових кроків. Також буде розглянуто процес генерації карти Random Walk з різними варіаціями параметрів. Це дозволить нам побачити, як зміна різних налаштувань впливає на кінцевий результат, і дасть можливість знайти оптимальні параметри для досягнення бажаного вигляду карти. На рисунку 3.9 зображено інспектор в якому наявні параметри що використовуватимуться для генерацій.

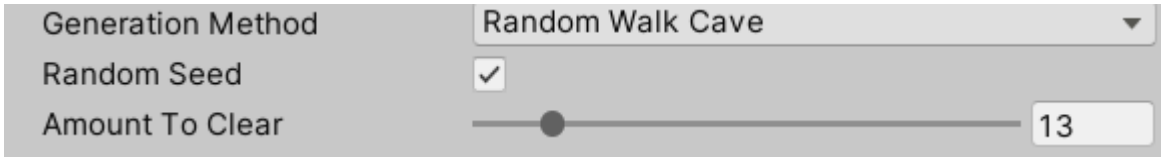

Рисунок 3.9 – Інспектор з параметрами для генерації за допомогою методу Random Walk

На рисунку зображено налаштування генерації карти методом Random Walk в інтерфейсі користувача. Ось опис основних параметрів:

– Random Seed (Випадкове зерно): прапорець, який визначає, чи буде використовуватися фіксоване випадкове зерно. Якщо прапорець встановлено, то генерація карти буде детермінованою, тобто однаковою при кожному запуску з тим самим зерном;

– Amount To Clear (Кількість для очищення): слайдер і поле введення, які визначають кількість клітинок, що будуть очищені в процесі генерації. Це впливає на те, наскільки простора буде карта.

Ці параметри дозволяють налаштувати процес генерації карти для досягнення різноманітних результатів, що можуть відповідати різним ігровим вимогам або естетичним уподобанням.

Результат генерації декількох варіантів карти зображено нижче на рисунку 3.10.

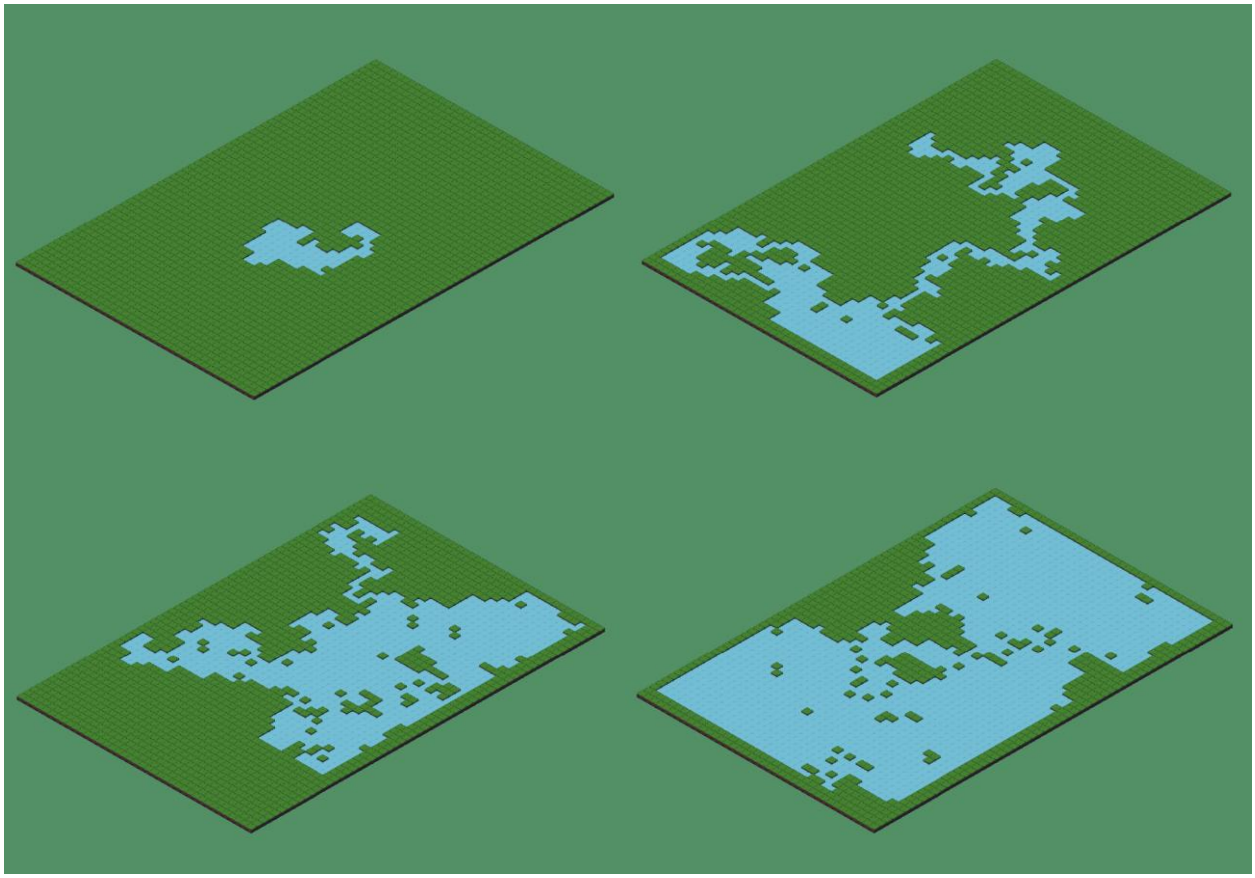

Рисунок 3.10 – Декілька варіантів генерації методом Random Walk

На основі чотирьох зображень, які демонструють різні генерації карт цим методом, можна зробити кілька висновків:

Розмір та форма водойм:

– на першому зображенні ми бачимо невелику водойму, яка має нерівну форму;

– на другому зображенні водойма вже значно більша і має довгі, викривлені форми, що наче утворюють річки та озера;

– третє зображення показує ще більш розгалужену структуру водойм, які покривають значну частину карти;

– четверте зображення демонструє практично повністю заповнену водою карту з лише кількома острівцями суші.

Еволюція карти:

– генерація карт методом Random Walk починається з малого і поступово розширюється;

– перші кроки генерації створюють невеликі водойми, які з часом зливаються і утворюють складніші структури;

Випадковість і хаотичність:

– усі чотири зображення показують, що метод Random Walk створює досить хаотичні та випадкові форми ландшафтів, що нагадують природні водойми;

– випадковість у формуванні водойм може бути корисною для створення унікальних та непередбачуваних ігрових світів.

Щодо масштабування - від першої генерації до четвертої видно, як поступово збільшується масштаб водойм, що може бути корисним при використанні цього методу для різних рівнів деталізації у грі чи додатку.

Можливості адаптації:

– метод Random Walk можна адаптувати для створення різних типів ландшафтів, від невеликих озер до великих морів та океанів;

– регулювання параметрів генерації (наприклад, кількість кроків, напрямок руху) дозволяє контролювати складність та масштаб отриманих карт.

Як висновок метод Random Walk є гнучким і ефективним інструментом для генерації випадкових карт. Він дозволяє створювати різноманітні та природні форми водойм, що може бути дуже корисним для розробників ігор та інших додатків, де потрібна випадкова генерація ландшафтів. Однак, для досягнення бажаних результатів може знадобитися налаштування параметрів генерації.

3.5 Розробка демонстраційного додатку

У цьому розділі буде детально описано процес створення демонстраційного додатку, який генерує 2D карту з використанням різних методів генерації в Unity. Цей додаток призначений для демонстрації можливостей процедурної генерації ландшафтів та елементів оточення у двовимірному просторі. Ми розглянемо етапи створення поверхні, додання дерев, додання декорацій та генерації річки. Кожен з цих етапів має свої особливості та методи реалізації, що забезпечують різноманітність та реалістичність створюваних карт.

Спочатку буде описано створення базової поверхні, яка слугуватиме основою для подальших елементів карти. Потім ми перейдемо до додання дерев, які додають природний вигляд та варіативність ландшафту. Після цього розглянемо додання декорацій, які можуть включати камені, кущі та інші об'єкти, що роблять карту більш деталізованою та цікавою для користувача. На завершення, ми розглянемо генерацію річки, яка додає динаміку та різноманітність у структуру карти.

У заключній частині розділу буде представлено демонстрацію результатів, де показано декілька варіантів генерації карти, що розрізняються за величиною.

Це дозволить оцінити гнучкість та ефективність розроблених методів процедурної генерації у різних умовах та масштабах.

3.5.1 Поетапна розробка генерації 2D карти

Одним із ключових етапів розробки процедурно згенерованої 2D карти є створення базової поверхні, яка слугуватиме основою для подальших елементів ландшафту. Для цього ми використовуємо метод Perlin Noise, який дозволяє створити реалістичну та детальну текстуру землі.

Метод Perlin Noise є популярним алгоритмом для генерації згладжених, природно виглядаючих шумових карт. Він створює плавні переходи між значеннями, що робить його ідеальним для створення таких природних ландшафтів, як гори, пагорби та долини. Основна ідея полягає в тому, що для кожної точки карти обчислюється значення шуму, яке визначає висоту або тип поверхні в цій точці.

У додатку було використано декілька тайлів для наглядності та підвищення деталізації. Процес генерації поверхні за допомогою Perlin Noise складається з наступних кроків:

Перший крок – це ініціалізація сітки. Створюється двовимірна сітка, яка представляє карту. Кожна клітинка сітки містить значення шуму, яке визначатиме тип поверхні в цій точці.

Далі використовується функція Perlin Noise для генерації шуму для кожної точки сітки. В залежності від значення шуму в кожній точці, було присвоюєно відповідний тайл. Наприклад: значення 0 - 0.7 – рівнинам, а занчення 0.7 - 1 – поля.

Отримані значення використовуються для відображення різних тайлів на карті, створюючи таким чином детальну та різноманітну поверхню.

Цей підхід дозволяє створювати багатошарові та реалістичні ландшафти, які можна легко налаштовувати за допомогою зміни параметрів Perlin Noise та

різних тайлів. Результат генерації буде мати вигляд природного ландшафту з плавними переходами між різними типами поверхонь.

Результат генерації поверхні зображений на рисунку 3.11.

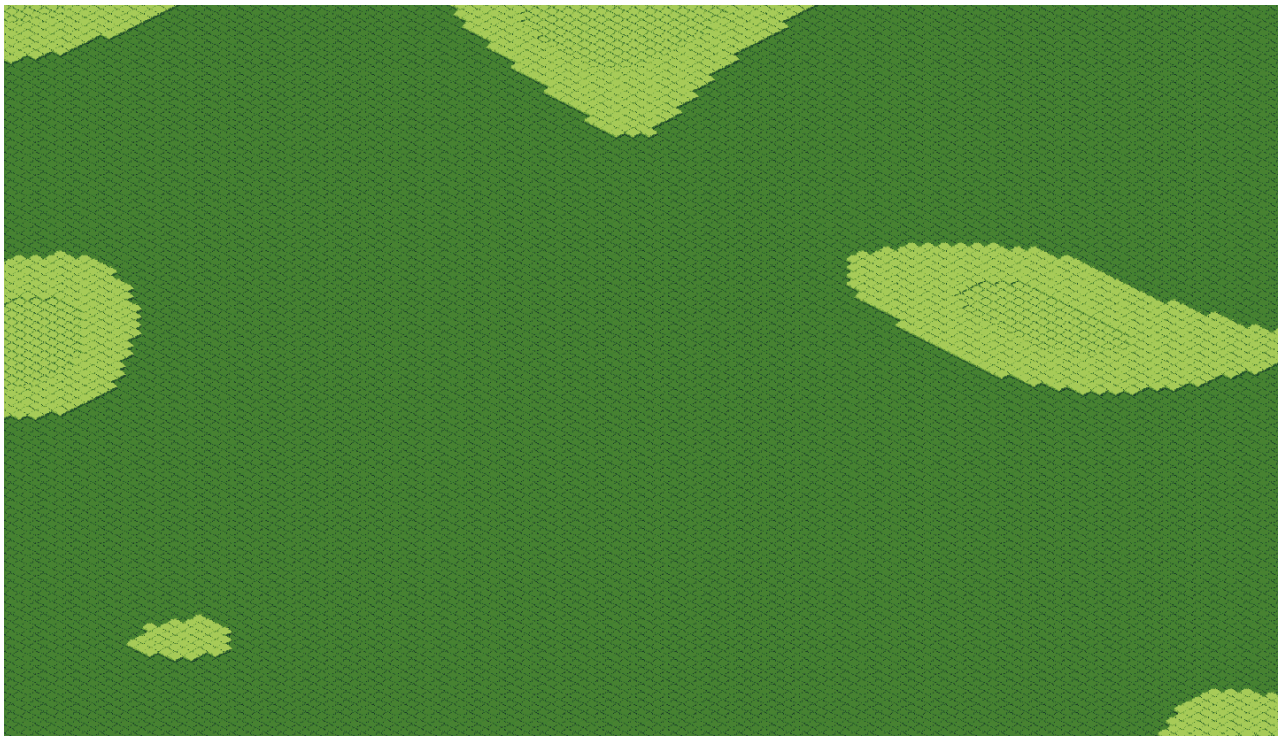

Рисунок 3.11 – Результат генерації поверхні методом Perlin noise

Після створення основи карти за допомогою методу Perlin Noise, наступним етапом є додавання декорацій, таких як квіти та трава, для підвищення візуальної привабливості ландшафту. Для цього було використано метод Random Walk, який дозволяє випадковим чином розмістити декорації по всій поверхні карти.

Метод Random Walk полягає у випадковому русі по сітці карти та розміщенні об'єктів на певних клітинках. Цей підхід створює природний вигляд розташування об'єктів, що особливо підходить для таких елементів, як квіти та трава, які зазвичай розташовані не систематично, а хаотично. Ось детальний опис процесу:

Спершу було обрано початкову точку на карті, з якої почнеться наш випадковий рух. Це може бути будь-яка клітинка сітки.

Випадковий рух: Від початкової точки ми починаємо випадковий рух по карті. На кожному кроці алгоритм випадковим чином вибирає напрямок руху: вгору, вниз, вліво або вправо.

На кожному кроці випадкового руху перевіряється, чи підходить поточна клітинка для розміщення декорацій (наприклад, чи не зайнята вона іншим об'єктом). Якщо клітинка підходить, на ній розміщується спрайт квітки або трави. Для наглядності використовується різні спрайти, щоб створити різноманітність в оформленні карти.

Повторюємо процес випадкового руху та розміщення декорацій до тих пір, поки не досягнемо потрібної кількості об'єктів на карті або поки не виконаємо задану кількість кроків.

Цей метод дозволяє нам створити природне та гармонійне розташування декорацій на карті, яке добре вписується в загальний вигляд ландшафту, згенерованого за допомогою Perlin Noise. Додавання квітів та трави додає деталізації та робить карту більш живою та реалістичною.

Таким чином, поєднання методів Perlin Noise та Random Walk дозволяє створювати багатошарові та детальні 2D карти з різноманітними елементами ландшафту, що підвищує їх візуальну привабливість та функціональність.

Результат генерації декорацій зображений нижче на рис 3.12.

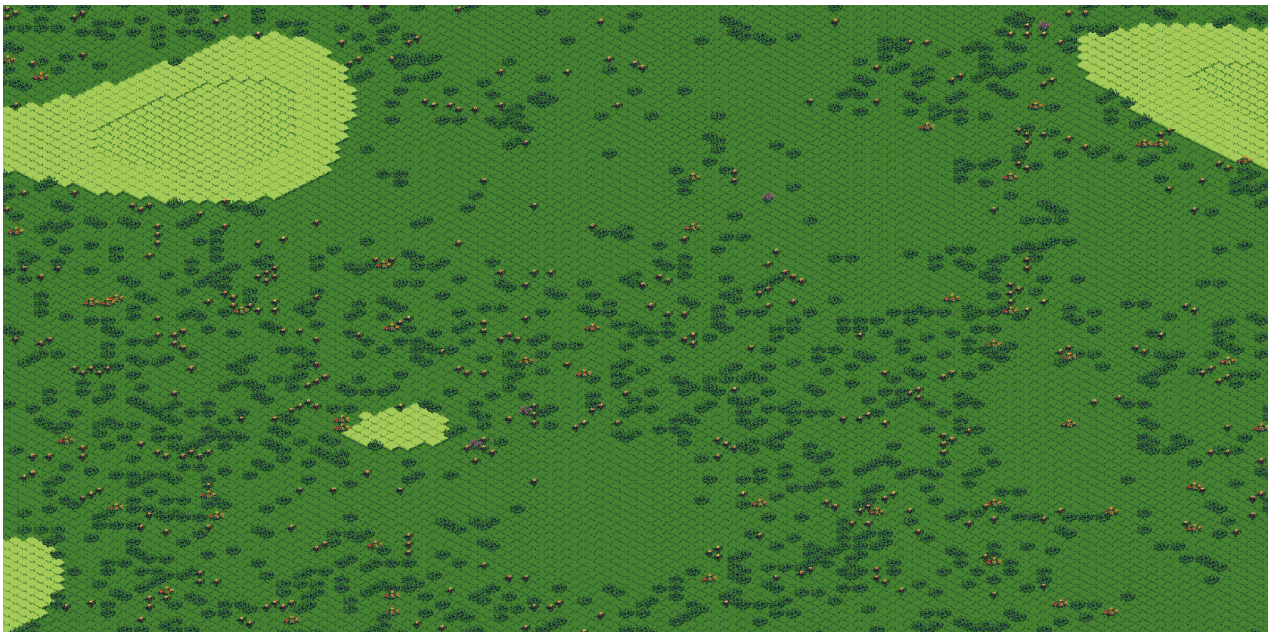

Рисунок 3.12 – Результат генерації декорацій методом Random walk

Після створення основи карти та додання декорацій, таких як квіти та трава, наступним етапом є генерація дерев на ландшафті. Для цього використовувався метод Cellular Automata, який дозволяє створювати складні структури на основі простих правил. Цей метод особливо добре підходить для генерації дерев, оскільки створює природний і випадковий розподіл об'єктів.

Процес генерації дерев складається з кількох етапів:

Спершу було створено двовимірну сітку, яка відповідає розміру карти. Кожна клітинка сітки може бути або порожньою, або містити дерево.

Далі було заповнено сітку деревами з певною ймовірністю. Це створює початковий випадковий розподіл дерев по карті.

Використовується правила Cellular Automata для визначення, чи повинна кожна клітинка містити дерево в наступному поколінні. Одним з популярних наборів правил є такі:

– клітинка стає деревом, якщо вона оточена більш ніж чотирма деревами;

– клітинка залишається деревом, якщо вона оточена трьома або більше деревами;

– в іншому випадку клітинка стає порожньою.

Згладжування методом Moore Neighbourhood: Для досягнення більш природного вигляду лісу, виконуємо кілька циклів згладжування. Метод Moore Neighbourhood передбачає врахування всіх восьми сусідніх клітинок навколо кожної клітинки. Це дозволяє усунути ізольовані дерева та групи дерев, створюючи більш злиту лісову структуру.

Для кожного дерева випадковим чином вибирається один з декількох спрайтів, що представляють різні види дерев. Це додає різноманіття та робить ліс більш реалістичним.

Після завершення циклів згладжування, остаточна сітка використовується для розміщення дерев на карті. Дерева розміщуються поверх тайлів основи, враховуючи їх координати.

Цей підхід дозволяє створювати природні лісові масиви з різноманітними деревами, які гармонійно вписуються в загальний ландшафт карти. Метод Cellular Automata у поєднанні зі згладжуванням за допомогою Moore Neighbourhood створює органічний розподіл дерев, що значно підвищує реалістичність та візуальну привабливість карти.

Таким чином, ми отримуємо багатошаровий ландшафт з різними елементами, такими як земля, декорації та дерева, створюючи детальну і реалістичну 2D карту.

Результат зображений нижче на рисунку 3.13.

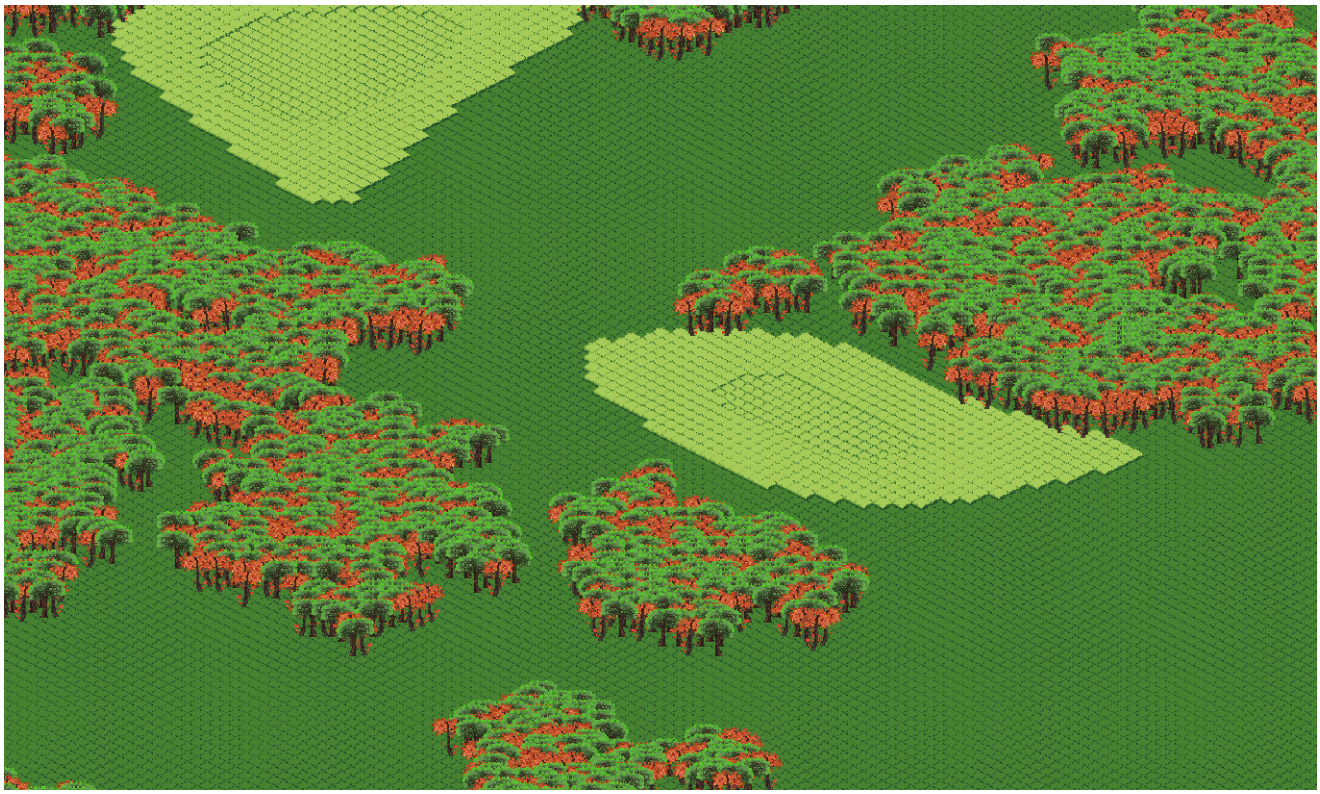

Рисунок 3.13 – Результат генерації дерев методом Cellular Automata та зглажування методом Moore Neighbourhood.

Процес генерації річки, яка проходить через усю карту, було реалізовано за допомогою методу Directional Tunnel. Цей метод передбачає створення плавного та природного шляху для річки, що дозволяє їй виглядати органічно в контексті загального ландшафту. Початково було задано стартову точку річки на одній з граней карти, після чого обрано напрямок руху, який змінюється під час генерації річки, щоб уникнути прямолінійності та створити більш природні вигини та повороти.

Генерація річки починається з вузького тунелю, який поступово розширюється та звужується, створюючи різноманітні секції. На кожному кроці було враховано можливість зміни напрямку, щоб річка могла огинати перешкоди та плавно виглядати на карті. Після побудови основного маршруту річки, було додано випадкові відгалуження та зміщення, щоб додати реалістичності. Важливо було забезпечити, щоб річка мала зв’язний потік і не створювала замкнених контурів.

Після того, як основний шлях річки було створено, проведено декілька циклів згладжування методом Moore Neighbourhood. Цей метод враховує сусідні клітинки навколо кожної точки річки, дозволяючи усунути різкі переходи та створити більш плавні контури. Завдяки цьому річка набуває природного вигляду та гармонійно вписується в загальний ландшафт карти. Згладжування також допомагає уникнути занадто вузьких або широких секцій, забезпечуючи стабільну ширину річки по всій її довжині.

В результаті застосування методу Directional Tunnel у поєднанні зі згладжуванням методом Moore Neighbourhood було створено реалістичну та органічну річкову систему, яка проходить через усю карту. Це додає динаміки та візуальної глибини 2D ландшафту, роблячи його більш живим та природним. Річка стає центральним елементом карти, навколо якого можна будувати інші ландшафтні особливості, такі як ліси, поля та гори, що підвищує загальну реалістичність та привабливість створеної карти.

Результат генерації річки зображени й нижче на рисунку 3.14.

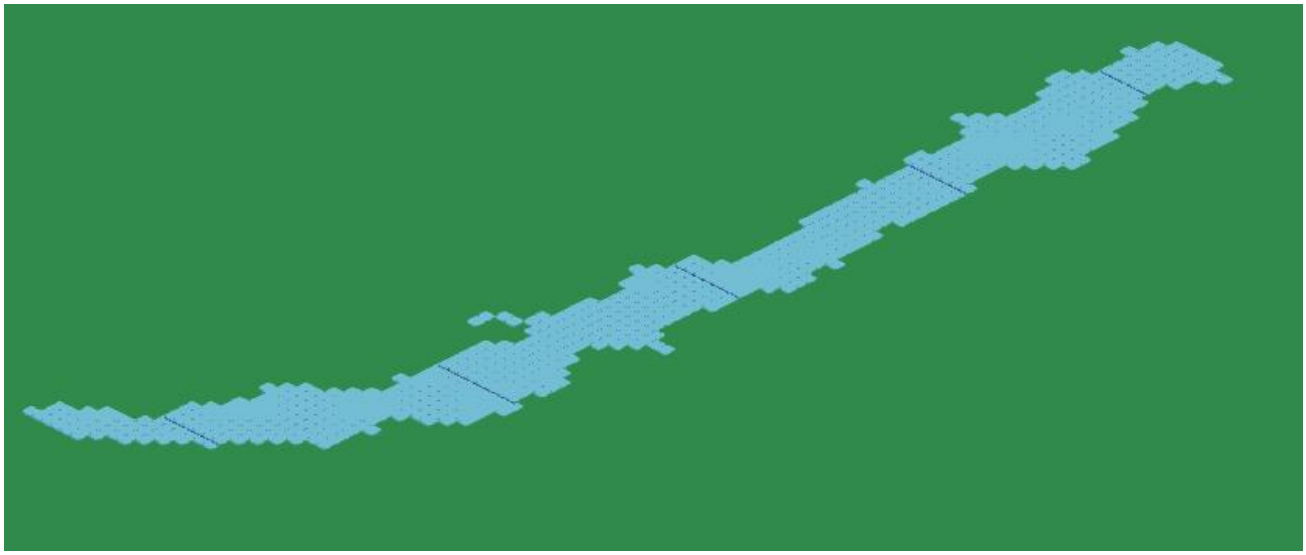

Рисунок 3.14 – Результат генерації річки методом Directional Tunnel та зглажування методом Moore Neighbourhood.

3.5.2 Презентація результатів

Було розроблено процес генерації двовимірної карти, в основі якого лежить використання методу Perlin noise для створення рельєфу та топографії. Цей метод дозволив отримати реалістичні ландшафтні форми, здатні відображати різноманітні характеристики території. Після цього було використано алгоритм Random walk для розміщення декоративних елементів, таких як трава та квіти, що призвело до природнього вигляду карти.

Далі, для збагачення ландшафту, було використано метод Cellular automata для генерації дерев та їхнього розміщення. Цей метод сприяв утворенню лісистих масивів та забезпечив більшу різноманітність та природність отриманої карти. Після розміщення дерев було застосовано алгоритм згладжування за допомогою методу Moore Neighbourhood для поліпшення їхнього вигляду та інтеграції в ландшафт.

Наступним етапом у процесі генерації було створення річок за допомогою методу Directional Tunnel, який дозволив моделювати природній хід руху води, та подальше їх згладжування методом Moore Neighbourhood для забезпечення більш природного вигляду та інтеграції з оточуючим ландшафтом. В результаті цих кроків було отримано карту з реалістичним ландшафтом, де всі елементи, включаючи ландшафтні форми, декоративний деталі, дерева та річки, були поєднані в єдину гармонійну композицію.

Результати генерацій зображені на рисунку 3.15.

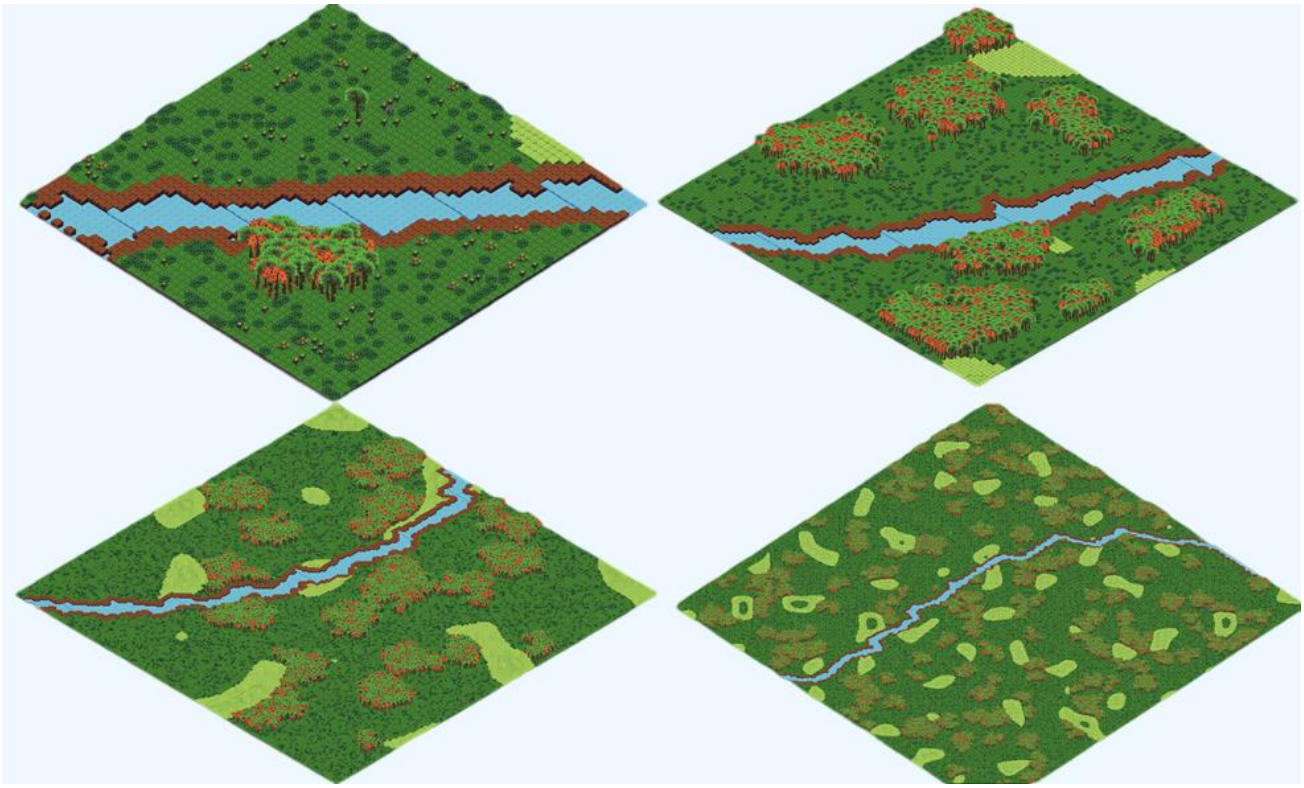

Рисунок 3.15 – Згенерована, різними методами, карта в різних розмірах

4 АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Вибір критеріїв для оцінки методів генерації

Далі будуть наведені критерії для порівняння методів генерації двовимірної карти, що дозволять оцінити їх ефективність та придатність для створення реалістичних та привабливих ландшафтів. Критерії будуть спрямовані на оцінку якості ландшафту, рівномірності розміщення об'єктів, природності форм та здатності інтеграції різних елементів, таких як дерева, трава та річки, у єдину гармонійну картину. Таке порівняння дозволить визначити найбільш підходящий метод для конкретних потреб генерації карти залежно від бажаних характеристик та естетичних вимог. До конкретних критеріїв для порівняння методів генерації 2D карт можна віднести наступні:

Різноманіття:

– на скільки різноманітні та унікальні карти, що генеруються методом;

– чи може метод генерувати карти different типів (наприклад, ліси, пустелі, гори);

– чи може метод генерувати карти з різними характеристиками (наприклад, розмір, форма, складність).

Реалізм:

– наскільки реалістичні карти, що генеруються методом;

– чи відповідають характеристики карт реальним географічним об'єктам;

– чи враховує метод закони фізики та інші фактори, що впливають на формування ландшафту?

Контроль:

– на скільки гнучко можна контролювати параметри генерації карт;

– чи можна задавати чіткі параметри для генерації карт з певними характеристиками;

– чи можна генерувати карти з візуальними деталями (наприклад, дороги, річки, будівлі)?

Ефективність:

– наскільки швидко метод генерує карти;

– чи можна використовувати метод для генерації великих карт;

– чи можна оптимізувати метод для покращення його продуктивності?

Простота використання:

– наскільки легко використовувати метод;

– чи доступні інструменти та бібліотеки для полегшення використання методу;

– чи добре документований метод?

Додаткові критерії:

– наскільки легко модифікувати метод для додавання нових функцій;

– чи є доступна спільнота або команда розробників, що підтримують метод.

4.2 Оцінка і аналіз результатів

Далі буде наведено оцінку та аналіз результатів для кожного з методів генерації двовимірної карти з урахуванням всіх раніше проведених етапів розробки. Цей завершальний аналіз зосереджений на оцінці якості кінцевої карти для кожного методу, враховуючи його ефективність, природність отриманих форм, а також гармонійне поєднання всіх елементів ландшафту. Під час аналізу будуть розглянуті переваги та недоліки кожного методу з метою визначення найбільш оптимального варіанту для конкретних потреб.

Метод Perlin Noise:

– різноманіття: метод може генерувати різні типи карт з високою унікальністю та характеристиками;

– реалізм: карти виглядають дуже реалістичними, особливо для природних ландшафтів;

– контроль: дуже гнучкий метод з можливістю налаштування параметрів;

– ефективність: швидкий метод, який можна використовувати для великих карт;

– простота використання: може бути складний для новачків, але доступні численні інструменти та бібліотеки;

– додаткові критерії: легко модифікувати, хороша підтримка спільноти.

Random Walk:

– різноманіття: унікальність карт середня, обмежений у варіантах типів карт;

– реалізм: менш реалістичні карти, не враховує закони фізики;

– контроль: низька гнучкість, важко задавати чіткі параметри;

– ефективність: дуже швидкий метод, підходить для великих карт;

– простота використання: легкий у використанні, але бракує інструментів і бібліотек.

– додаткові критерії: легко модифікувати, але низька підтримка.

Метод Directional Move:

– різноманіття: висока унікальність та можливість генерувати різні типи карт;

– реалізм: високий рівень реалізму, частково враховує закони фізики;

– контроль: гнучкість параметрів висока, можливість задавати чіткі параметри та генерувати візуальні деталі;

– ефективність: швидкість генерації середня, підходить для великих карт, можливість оптимізації;

– простота використання: може бути складним для новачків, доступні деякі інструменти та бібліотеки, середня документація;

– додаткові критерії: легко модифікувати, середня підтримка.

Метод Cellular Automata (Moore Neighborhood):

– різноманіття: середня унікальність та можливість генерування різних типів карт з різними характеристиками;

– реалізм: середній рівень реалістичності, частково відповідає реальним об'єктам, не враховує закони фізики;

– контроль: середня гнучкість параметрів, можливість задавати чіткі параметри;

– ефективність: висока швидкість генерації, підходить для великих карт, можливість оптимізації;

– простота використання: легкий у використанні, доступні інструменти та бібліотеки, хороша документація;

– додаткові критерії: легко модифікувати, хороша підтримка.

Метод Cellular Automata (Von Neumann Neighborhood):

– різноманіття: середня унікальність та можливість генерування різних типів карт з різними характеристиками;

– реалізм: середній рівень реалістичності, частково відповідає реальним об'єктам, не враховує закони фізики;

– контроль: середня гнучкість параметрів, можливість задавати чіткі параметри;

– ефективність: висока швидкість генерації, підходить для великих карт, можливість оптимізації;

– простота використання: легкий у використанні, доступні інструменти та бібліотеки, хороша документація;

– додаткові критерії: легко модифікувати, хороша підтримка.

Таким чином зібравши результати можно представити їх у вигляді таблиці, яка представлена нижче.

Таблиця 4.1 – Порівняння методів генерації

Критерій	**Perlin Noise**	**Random Walk**	**Directional Move**	**Cellar Automata (Moore Neighborhood)**	**Cellar Automata (Von Neumann Neighborhood)**
Різноманіття					
Унікальність карт	Висока	Середня	Висока	Середня	Середня
Різні типи карт	Так	Обмежено	Так	Так	Так
Різні характеристики	Так	Так	Так	Так	Так
Реалізм					
Реалістичність	Висока	Середня	Висока	Середня	Середня
Відповідність реальним об'єктам	Так	Частково	Так	Частково	Частково
Закони фізики	Частково	Ні	Частково	Ні	Ні
Контроль					
Гнучкість параметрів	Висока	Низька	Висока	Середня	Середня
Чіткі параметри	Так	Ні	Так	Так	Так
Візуальні деталі	Обмежено	Ні	Так	Обмежено	Обмежено
Ефективність					

Критерій	**Perlin Noise**	**Random Walk**	**Directional Move**	**Cellar Automata (Moore Neighborhood)**	**Cellar Automata (Von Neumann Neighborhood)**
Швидкість генерації	Висока	Висока	Середня	Висока	Висока
Генерація великих карт	Так	Так	Так	Так	Так
Оптимізація	Так	Так	Так	Так	Так
Простота використання					
Легкість використання	Середня	Висока	Середня	Висока	Висока
Інструменти та бібліотеки	Так	Ні	Так	Так	Так
Документація	Висока	Низька	Середня	Висока	Висока
Додаткові критерії					
Модифікація	Середня	Висока	Середня	Висока	Висока
Підтримка	Висока	Низька	Середня	Висока	Висока

В результаті можна підвести загальні висновки:

– Perlin Noise є найбільш реалістичним та гнучким методом, але може бути складним для початківців;

– Random Walk є простим у використанні та швидким, але менш реалістичним та контрольованим;

– Directional Move забезпечує високу унікальність та гнучкість, але може вимагати більше часу на генерацію;

– Cellular Automata (Moore Neighborhood) і Cellular Automata (Von Neumann Neighborhood) пропонують хорошу швидкість та простоту використання, але менш реалістичні та гнучкі в порівнянні з іншими методами.

ВИСНОВОК

У цій роботі було розглянуто та проаналізовано різні методи генерації 2D карт місцевості для використання у відеоіграх. Проведено дослідження методів таких як Perlin noise, Directional tunnel, Random walk та Cellar automaton, а також їх комбінацій у сполученні з різними методами згладжування, такими як Moor Neighborhood та Von Neumann Neighborhood.

На основі досліджень було розроблено два додатки для Unity. Перший демонструє роботу кожного методу окремо, дозволяючи оцінити їхню ефективність та естетичність. Другий додаток відображає взаємодію всіх методів у вигляді повністю згенерованої карти, де ландшафт генерується за допомогою Perlin noise, декорації - Random walk, дерева - Cellar automaton з наступним згладжуванням за допомогою методу Moor Neighborhood та річки - Directional tunnel.

В ході досліджень було проведено порівняльний аналіз кожного методу з використанням певних критеріїв, таких як ефективність, природність генерованих форм, гармонійне поєднання елементів та загальна естетика карти. Результати цього аналізу були відображені у таблиці порівнянь, що дозволило визначити переваги та недоліки кожного методу та їхню відповідність вимогам конкретного використання.

У загальному, ця робота не лише дослідила різні методи генерації 2D карт місцевості, але й представила практичні додатки, які відображають їхню роботу у реальному часі. Такий підхід дозволив отримати глибоке розуміння переваг та обмежень кожного методу та забезпечити підґрунтя для подальших досліджень та вдосконалення процедурної генерації карт у відеоіграх.

Також за цією темою було підготовлено дві доповіді на конференціях: Харківський молодіжний форум та КНУКіМ.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ ТА ДЖЕРЕЛ ПОСИЛАННЯ

1. Tanya X. Short, Tarn Adams. Procedural Generation in Game Design, 2017. – 270c.

2. Ryan Watkins. Procedural Generation in Unity, 2019. – 246c.

3. David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, Steven Worley. Texturing & Modeling: A Procedural Approach. Morgan Kaufmann Series in Computer Graphics and Geometric Modeling, 2002. – 688c.

4. Daniel Shiffman. The Nature of Code: Simulating Natural Systems with Processing, 2012. – 520c.

5. Ryan Watkins. Procedural Generation in Unity, 2019. – 246c.

6. Sebastian Lague. Procedural Generation in Game Development, 2018. – 320c.

7. Jason Gregory. Game Engine Architecture, Third Edition, 2018. – 1248c.

8. Steve Rabin. AI Game Programming Wisdom, 2002. – 656c.

9. Penny de Byl. Procedural Generation in Unity Game Development, 2017. – 346c.

10. Dan Violet Sagmiller. Procedural Content Generation for Unity Game Development, 2016. – 368c.

11. Adam M. Smith, Michael Mateas. "A Case Study of Player Experience in Educational Games: Procedural Terrain Generation for Games." Proceedings of the FDG Workshop on Procedural Content Generation in Games. 2010.

12. Miguel Cepero. "Procedural Planet Generation." Gamasutra, 2012.

13. Eric Galin, Eric Guérin, Adrien Peytavie, Bedřich Beneš, Eric Heitz. "Real-Time Rendering, Third Edition." A K Peters/CRC Press, 2008.

14. Michael F. Deering. "Geometry Compression." Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques. ACM, 1995.

15. Stefan Freyr Gudmundsson, Michael C. Hansen, Rasmus R. Jensen, Mikkel G. Christensen, Jonas S. Nielsen. "Procedural Content Generation in Games:

A Textbook and an Overview of Current Research." Proceedings of the 2014 Annual Symposium on Computer-Human Interaction in Play. ACM, 2014.

16. Ben Garney. "Procedural Content Generation in Games." CRC Press, 2017.

17. Inigo Quilez. "Articles on Procedural Content Generation." [Online]. Available: http://www.iquilezles.org/. [Accessed: Jan. 15, 2024].

18. Jeff Bucklew. "Procedural Generation in Game Design: A Primer." CRC Press, 2020.

19. Michael Kissner. "Procedural Content Generation in Unity." [Online]. Available: https://catlikecoding.com/unity/tutorials/procedural-grid/. [Accessed: Jan. 15, 2024].

20. Damian Isla. "A Procedural Approach to Landscape Generation." Gamasutra, 2004.

21. Joaquim Felipó. "Procedural Content Generation in 2D Tile-Based Games." CRC Press, 2021.

22. Anton Gerdelan. "Real-Time Rendering, Fourth Edition." CRC Press, 2018.

23. Johannes Kopf, Michael F. Cohen, Dani Lischinski. "Terrain Rendering and Simulation." Morgan & Claypool, 2010.

24. Matt Fairclough. "Procedural Generation in Games: A Practical Guide." Apress, 2017.

25. Mark H. Overmars. "The Algorithmic Beauty of Plants." Springer, 2004.