ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Метод процедурної генерації об’єктів ігрового світу в жанрі “Tower Defense” на основі шумів»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Денис КОСЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-62
Денис КОСЕНКО

Керівник: Віктор ГРЕБЕНЮК
доктор філософії (PhD)

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Косенко Денису Максимовичу

1. Тема кваліфікаційної роботи: «Метод процедурної генерації об’єктів ігрового світу в жанрі “Tower Defense” на основі шумів»

керівник кваліфікаційної роботи Віктор ГРЕБЕНЮК, доктор філософії (PhD),

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» жовтня 2024 р. № 320.

2. Строк подання кваліфікаційної роботи «17» грудня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, методи генерації об’єктів ігрового світу.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Дослідження генерації об’єктів ігрового світу в іграх жанру Tower Defense
2. Аналіз методів генерації об’єктів ігрового світу в грі жанру Tower Defense.
3. Розробка вимог до власного методу генерації ігрових об’єктів.

5. Перелік ілюстративного матеріалу: *презентація*

1. Мета, об’єкт, предмет дослідження.
2. Порівняльна характеристика існуючих методів.
3. Математична модель процедурної генерації
4. Математична модель розміщення об'єктів
5. Блок схема методу
6. Результат процедурної генерації

6. Дата видачі завдання «16» жовтня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	16.10-23.10.2024	
2	Аналіз існуючих методів вирішення поставленої задачі.	24.10-28.10.2024	
3	Дослідження методів процедурної генерації	29.10-04.11.2024	
4	Аналіз особливостей впливу методів процедурної генерації на створення ігрового світу	05.11-08.11.2024	
5	Розробка програмного забезпечення для створення власного методу генерації об’єктів для гри в жанрі Tower Defense.	09.11-19.11.2024	
6	Застосування власного методу в програмному забезпеченні для генерації об’єктів для гри в жанрі Tower Defense.	20.11-23.11.2024	
7	Оформлення роботи: вступ, висновки, реферат	24.11-27.11.2024	
8	Розробка демонстраційних матеріалів	28.11-29.11.2024	
9	Попередній захист роботи	30.11-12.12.2024	

Здобувач вищої освіти _____ Денис КОСЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Віктор ГРЕБЕНЮК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 57 стор., 1 табл., 23 рис., 20 джерел.

Мета роботи – спрощення процесу створення ігрових карт із використанням шумових функцій та алгоритмів розміщення.

Об’єкт дослідження – процес процедурної генерації та розміщення об'єктів оточення в іграх.

Предмет дослідження – методи процедурної генерації ігрових карт і алгоритми розміщення об'єктів оточення.

Досліджено різноманітні підходи до генерації об'єктів, такі як випадкове розташування, використання шуму Перліна, генератора псевдовипадкових чисел та інші. Проаналізовано переваги та недоліки кожного методу з метою розробки ефективного алгоритму генерації, спрямованого на покращення геймплею та реіграбельності у віртуальному середовищі.

КЛЮЧОВІ СЛОВА: ПРОЦЕДУРНА ГЕНЕРАЦІЯ, ШУМ ПЕРЛІНА, TOWER DEFENCE, МОДУЛЬНА ГЕНЕРАЦІЯ, ОБ'ЄКТИ

ABSTRACT

Text part of the master's qualification work: 57 pages, 23pictures, 1 table, 20 sources.

The purpose of the work– simplification of the process of creating game maps using noise functions and placement algorithms.

Object of research – the process of procedural generation and placement of environment objects in games.

Subject of research – methods of procedural generation of game maps and algorithms for placing objects in the environment.

Summary of the work: Various approaches to the generation of objects, such as random location, use of Perlin noise, pseudorandom number generator, and others, have been studied. The advantages and disadvantages of each method were analyzed in order to develop an effective generation algorithm aimed at improving gameplay and replayability in a virtual environment.

KEYWORDS: PROCEDURAL GENERATION, PEARLIN NOISE, TOWER DEFENSE, MODULAR GENERATION, OBJECTS

ЗМІСТ

ВСТУП

Ігри у жанрі Tower Defense продовжують користуватися популярністю серед гравців завдяки стратегічному підходу до організації оборони та постійно змінним умовам, які вимагають від гравців адаптації та використання різних тактик. Важливим аспектом, який впливає на різноманітність ігрового досвіду, є унікальність ігрових карт і об’єктів навколишнього середовища, які створюють головний виклик для гравців. Проте створення таких карт вручну потребує значних витрат часу та ресурсів.

У сучасних умовах розробка ефективного методу процедурної генерації карт стає надзвичайно актуальною у зв'язку з підвищеними очікуваннями гравців і необхідністю забезпечення високої реіграбельності. Використання процедурних методів генерації, таких як карти шумів та модульна генерація, дозволяє автоматично створювати різні ігрові світи, кожен з яких буде унікальним і вимагатиме від гравців розробки нових стратегій.

Використання в роботі таких підходів, як шумова генерація та алгоритми розміщення об’єктів середовища, відкриває перспективи створення динамічних та непередбачуваних ігрових карт. Це збільшить інтерес до ігор Tower Defense, підвищить їх конкурентоспроможність на ринку відеоігор і залучить більше гравців.

Актуальність теми обґрунтовується наступними факторами:

- Зростання очікувань гравців. Сучасні гравці очікують унікальних вражень від ігор, які відрізняються від рівня до рівня. Автоматична генерація карт забезпечить різноманітність, зберігаючи високий інтерес до ігрового процесу.
- Оптимізація розвитку. Процедурна генерація контенту значно скорочує час і витрати на створення нових рівнів. Це особливо важливо для незалежних розробників і невеликих студій, які обмежені в ресурсах.
- Конкуренція на ринку відеоігор. У висококонкурентному середовищі розробники повинні пропонувати інноваційні рішення, які виділяють їхній продукт

серед інших. Процедурна генерація дозволяє створити унікальний ігровий мінімальними витратами на розробку.

Мета роботи: спрощення процесу створення ігрових карт із використанням шумових функцій та алгоритмів розміщення.

Об'єкт дослідження: процес процедурної генерації та розміщення об'єктів оточення в іграх.

Предмет дослідження: методи процедурної генерації ігрових карт і алгоритми розміщення об'єктів оточення.

У процесі виконання магістерської роботи були використані наступні методи дослідження:

Теоретичні методи

- Аналіз літературних джерел з теми дослідження
- Аналіз існуючих методів та засобів генерації об'єктів ігрового світу

На першому етапі дослідження було отримано теоретичні відомості з теми генерації ігрових об'єктів. Для цього було проаналізовано літературні джерела з цієї тематики. Зокрема, було вивчено наступні питання:

- Загальні принципи розміщення та генерації ігрових об'єктів
- Методологія генерації об'єктів
- Існуючі методи та засоби підвищення якості генерації ігрових об'єктів

На другому етапі дослідження було проведено експериментальне дослідження якості розробленого методу процедурної генерації. Основна мета цього етапу полягала у створенні генератора, здатного формувати ігрові карти з об'єктами різної складності. Після реалізації алгоритму було проведено серію експериментів для оцінки його ефективності.

Експерименти виконувалися за наступним планом:

Розроблений метод був протестований на наборах параметрів, які відображали умови різних сценаріїв гри.

Було згенеровано ігрові карти за допомогою поєднаного методу, результати яких порівнювалися з результатами інших підходів.

Проведено аналіз варіативності об’єктів та їх розташування на ігрових картах. Особливу увагу приділено органічності та реалістичності отриманих результатів.

Третій етап дослідження був присвячений інтеграції розробленого методу в прототип гри. Цей етап включав не лише технічну реалізацію, а й проведення масштабного тестування, яке дозволило оцінити адаптивність алгоритму до різних розмірів та структур карт. Особливу увагу було приділено продуктивності алгоритму, його здатності швидко та якісно генерувати ігровий світ навіть у випадках великих карт. У процесі тестування вдалося виявити потенційні недоліки, які могли впливати на якість генерації, та внести необхідні корективи. Оптимізація методу дозволила досягти вищої стабільності роботи алгоритму і підвищити загальну якість отриманих результатів.

Унікальність проведеного дослідження полягає у використанні комбінованого підходу, який поєднує метод шуму Перліна та лінійний конгруентний метод. Така комбінація дала змогу створювати карти, що відзначаються підвищеним рівнем деталізації, значною різноманітністю та природною структурною гармонією. Генеровані світи виглядають не лише реалістично, а й естетично привабливо, що є важливим аспектом для покращення загального ігрового досвіду.

Результати проведених експериментів підтвердили, що розроблений метод значно перевершує традиційні підходи. Він забезпечує краще та більш збалансоване розташування об'єктів, підвищену реалістичність ігрового середовища та оптимальний баланс елементів, необхідний для створення якісного геймплею. Такий підхід має потенціал для широкого використання у розробці ігор різних жанрів.

1 АНАЛІЗ АКТУАЛЬНОСТІ ТА ОГЛЯД ІСНУЮЧИХ МЕТОДІВ

1.1. Процедурна генерація як засіб підвищення ефективності розробки

Сучасна ігрова індустрія розвивається високими темпами, і з кожним роком вимоги до якості, різноманітності та масштабу ігрових проектів зростають. Це стосується як графічної складової ігор, так і глибини ігрового процесу, різноманітності механік і рівнів взаємодії гравців з ігровим світом.

Однією з головних проблем для розробників є необхідність створення унікальних, масштабних і різноманітних ігрових світів, які забезпечують занурення в гру на багато годин, але не вимагають величезних часових і фінансових витрат на їх ручну розробку.

Жанр Tower Defense (TD) — один з популярних напрямків у розробці ігор, яке характеризується простою механікою та глибоким геймплеєм. Унікальність ігрових світів і рівнів є ключовим фактором, що впливає на відтворюваність таких ігор. Традиційний підхід до створення рівня вимагає значних зусиль від дизайнерів і програмістів, що робить процес розробки дорогим і трудомістким.

У той же час технології процедурної генерації дозволяють автоматизувати частину роботи шляхом створення рівнів і об'єктів за допомогою алгоритмів, що особливо важливо в умовах обмежених бюджетів і необхідності швидкого прототипування.

Основою сучасної тривимірної графіки є математичні поняття. Визначення поверхні в тривимірному просторі неможливо без знання геометрії і володіння функціями. Проте сучасні програми, такі як 3ds Max, Cinema 4D, Blender, ZBrush, Autodesk Maya, Unreal Engine, Houdini, дозволяють створювати 3D-зображення без виконання складних математичних розрахунків вручну.

Використання процедурної генерації в жанрі Tower Defence дає розробникам можливість створювати рівні, які адаптуються до різних стилів гри, підвищують можливість відтворення та створюють більш різноманітний ігровий процес.

Важливою особливістю є можливість генерувати такі рівні на льоту, що дозволяє гравцям щоразу стикатися з новим ігровим світом, що є критичним для успіху таких проектів, як roguelike і tower defense.

Крім того, зі зростанням інтересу до інді-розробки ігор і збільшенням кількості невеликих студій використання процедурних методів стає все більш актуальним. Усе завдяки тому, що такі методи можуть значно скоротити витрати на розробку, уникаючи необхідності вручну проектувати кожен елемент світу.

Одним із найефективніших методів процедурної генерації є використання шумових функцій, таких як шум Перліна, симплексний шум та їх похідні. Шуми - це математичні функції, які генерують псевдовипадкові значення, що використовуються для створення природних ландшафтів, об'єктів і текстур. Ці методи широко використовуються в розробці ігор, оскільки вони дозволяють створювати плавні та деталізовані структури без артефактів або візуальної неоднорідності.

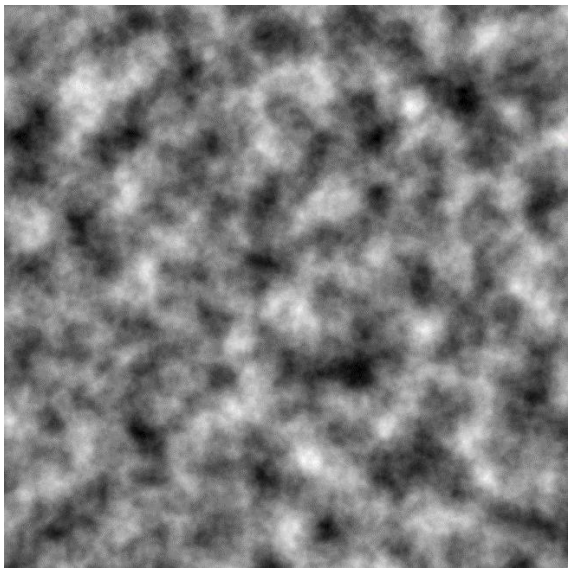


Рис. 1.1 Шум Перліна (Perlin Noise)

Шум Перліна, наприклад, спочатку був розроблений для створення реалістичних текстур у комп’ютерній графіці, але його використання значно розширилося до таких областей, як створення карти світу, створення об’єктів і реалізація динамічних ландшафтів у реальному часі. У контексті Tower Defense

використання шуму для процедурної генерації може надати можливість створювати нелінійні та асиметричні рівні, що сприяє різноманітності ігрового процесу та додає елемент непередбачуваності.

Процедурна генерація на основі шуму також є гнучким і розширюваним рішенням. Алгоритми можна адаптувати до різних типів об'єктів - від місцевості та доріг до структури баз і будівель, які можуть бути ключовими елементами стратегічного планування гри Tower Defense.

Цей метод ефективно вирішує одну з ключових проблем сучасних ігор – забезпечення високого рівня реіграбельності. Це особливо важливо в жанрі Tower Defense, де від гравців очікується проходження рівнів і опанування нових стратегій. Оскільки кожен процедурно згенерований рівень унікальний і пропонує нові виклики, гравець не відчуває себе одноманітним, що дозволяє йому тривалий час підтримувати інтерес до гри.

Замість традиційних сценаріїв, коли гравець вивчає конкретні рішення для конкретних ситуацій, процедурна генерація змушує його постійно шукати нові шляхи перемоги, що значно збільшує глибину ігрового процесу.

Оригінальність кожного ігрового рівня дозволяє гравцям зіткнутися з різними варіаціями ландшафтів, розташування ворожих баз, шляхів пересування противника та інших елементів. Це не тільки заохочує гравця адаптувати свою тактику до нових умов, але й розвиває навички стратегічного мислення, оскільки кожен новий рівень може вимагати іншого підходу для досягнення успіху.

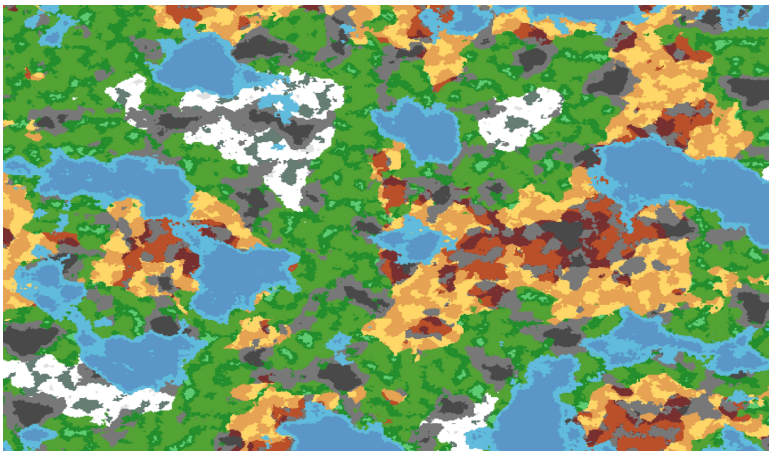


Рис 1.2 Приклад процедурно згенерованої мапи

Також процедурна генерація може враховувати стиль гри конкретного користувача. Наприклад, алгоритми можуть підлаштовуватися під рівень складності гравця, ігровий досвід і вподобання, забезпечуючи персоналізовані рівні та унікальні завдання для кожного.

Одним із найважливіших аспектів актуальності використання методів процедурної генерації є економія ресурсів на етапі розробки. Створення об'єктів і рівнів вручну вимагає багато часу і людських ресурсів, особливо в сучасних іграх з високим рівнем деталізації. Процедурна генерація дозволяє автоматизувати рутинні процеси, залишаючи дизайнерам і розробникам більше часу для роботи над унікальними елементами ігрового процесу та створення інноваційної механіки.

Алгоритми генерації на основі шуму вимагають мінімального налаштування та можуть бути легко масштабовані залежно від потреб проекту. Це особливо важливо для невеликих студій і незалежних розробників, які часто працюють з обмеженим бюджетом і часом. Таким чином, використання процедурної генерації об'єктів і рівнів стає невід'ємною частиною сучасної розробки ігор.

Одним із успішних прикладів гри у жанрі Tower Defense із використанням процедурної генерації є «Rogue Tower». У цій грі процедурна генерація використовується для створення унікальних карт зі змінною структурою шляхів, по яких рухаються хвилі ворогів. На відміну від класичних статичних рівнів, кожна ігрова сесія пропонує гравцеві новий, раніше небачений маршрут, що робить кожну гру унікальною та динамічною.

Основним методом процедурної генерації в Rogue Tower є алгоритм зростання карти. На кожному етапі гри карта розширюється, додаючи нові сегменти, які формуються за допомогою алгоритмів випадкового вибору та розгалуження. Цей процес забезпечує непередбачуваність розташування шляхів ворога до центральної вежі гравця. Нові розділи карти з’являються послідовно, змушуючи гравця швидко адаптуватися до мінливої структури маршруту та розподіляти ресурси для будівництва нових оборонних веж.

Отже, актуальність розробки методу процедурної генерації визначається загальними тенденціями розвитку ігрової індустрії, необхідністю створення

унікальних та адаптивних ігрових світів, підвищення реіграбельності та економія ресурсів розробника. Використання функцій шуму в цьому контексті дозволяє ефективно вирішувати проблему створення складних і різноманітних рівнів гри, забезпечуючи високу якість ігрового процесу та персоналізований ігровий досвід для користувачів.

1.2. Огляд аналогів та існуючих рішень

Аналоги та існуючі рішення в області процедурної генерації ігрових рівнів і контенту є важливим елементом для розуміння місця і новизни методу, що розробляється. Вивчення існуючих підходів дозволяє побачити як сильні, так і слабкі сторони поточних рішень, визначити, як можна вдосконалити алгоритми, а також зрозуміти, які інновації вже успішно реалізовані. У сфері ігор у жанрі Tower Defense процедурна генерація використовується рідше, ніж у таких жанрах, як Roguelike чи Open World, але є безліч прикладів успішних ігор, у яких використовуються аналогічні методи створення контенту.

Одним із найпоширеніших жанрів, де процедурна генерація знайшла широке застосування, є Roguelike. Ігри цього жанру характеризуються високою реграбельністю, де кожне нове проходження гри пропонує унікальні умови, завдяки яким гравці з кожною сесією отримують нові враження. Наприклад, “The Binding of Isaac” використовує процедурну генерацію для створення рандомізованих підземель, де гравець стикається з новими комбінаціями ворогів, об’єктів і перешкод, значно підвищуючи можливість повторної гри та захоплюючи гравців годинами.

Процедурна генерація в The Binding of Isaac охоплює кілька ключових елементів гри: рівні, кімнати, ворогів, предмети та події. Кожен ігровий сеанс починається зі створення унікального початкового значення, яке служить основою для створення всього ігрового світу. Seed визначає початкові умови та параметри, за допомогою яких процедурні алгоритми формують ігрове середовище. Важливо зазначити, що хоча генерація в грі є випадковою, вона підпорядковується суворим

правилам і обмеженням, які забезпечують збалансований ігровий процес і логічний прогрес складності.

Ігровий світ Binding of Isaac складається з багатьох поверхів, які є рівнями з певною структурою та набором кімнат. Кожен поверх генерується процедурно та включає обов’язкові елементи, такі як стартова кімната, кімната для боротьби з босами, секретні кімнати, магазин та інші спеціальні зони. Рівні в грі побудовані за принципом графа, де кімнати з'єднані між собою дверима і коридорами.

На кожному поверсі є кілька зон, кожна з яких містить набір приміщень, які сполучені між собою. Процес генерації кімнат починається зі створення карти поверху, де ключові кімнати (такі як кімната боса та магазин) спочатку розміщуються на карті, а потім простір, що залишився, заповнюється випадковими кімнатами із заздалегідь підготовленого пулу. Такий підхід дозволяє поєднувати випадковість і заздалегідь визначені елементи, що допомагає зберегти логічну структуру рівнів і їх узгодженість.

Кожна кімната в The Binding of Isaac також генерується процедурно. Кімнати можуть бути різних типів. Для створення кімнат у грі використовується система готових шаблонів кімнат, які потім випадковим чином заповнюються об’єктами, ворогами та предметами. Внутрішню структуру приміщення створюють на основі кількох параметрів: розміру приміщення, типу приміщення, складності поверху. Кожне рішення щодо розміщення ворогів, пасток і предметів у кімнаті підпорядковується певним правилам, щоб забезпечити збалансований рівень складності. Процедурна генерація кімнат базується на системі тегів і категорій, що дозволяє грі вибирати відповідні елементи для кожної кімнати залежно від її призначення та контексту.

В іграх з відкритим світом, таких як Minecraft, процедурна генерація є фундаментальною частиною ігрового процесу. Світ генерується на основі різноманітних шумових функцій (шум Перліна, симплексний шум тощо), що дозволяє створювати різноманітні ландшафти, біоми, споруди та підземелля. Такий підхід дозволяє гравцям досліджувати величезні території, кожна з яких має унікальні особливості, що надає грі високий рівень свободи та варіативності.

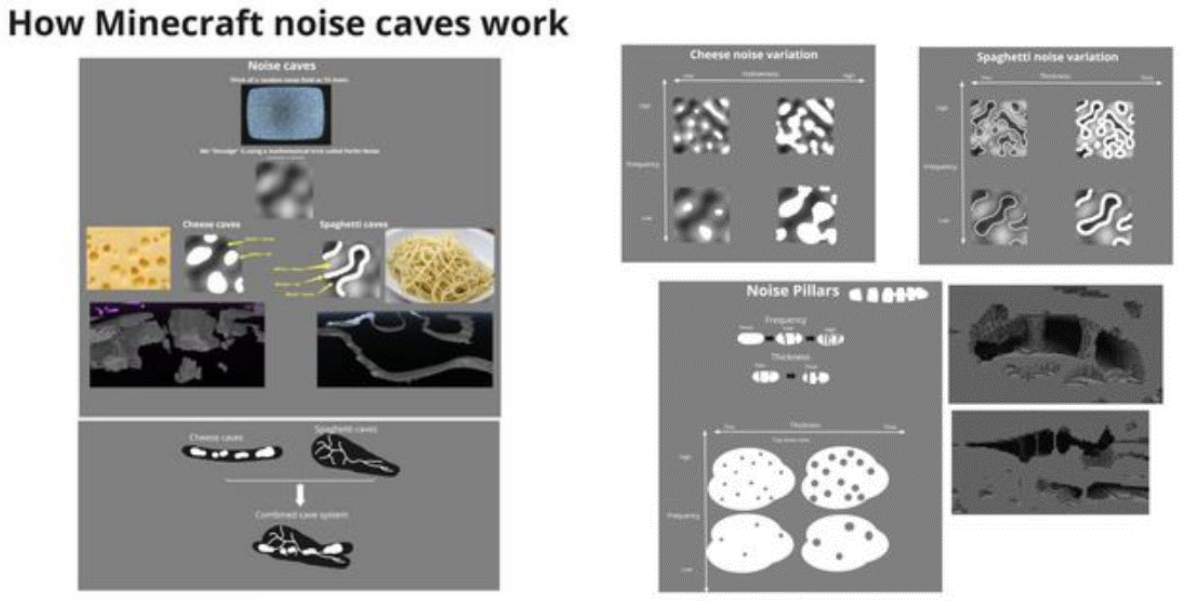


Рис 1.3 Використання шуму для генерації печер у Minecraft

Основною метою процедурної генерації є створення унікальних ігрових світів кожного разу, коли гравець починає нову гру. Генерація здійснюється за допомогою псевдовипадкових чисел і різноманітних шумових алгоритмів, що забезпечує як унікальність кожного світу, так і структурну логіку його елементів. Ігровий світ генерується на основі початкових значень, які є числовими значеннями, які використовуються для ініціалізації процедурної генерації. Це дозволяє гравцям відтворювати один і той самий світ знову і знову, якщо вони знають його насіння, або створювати абсолютно нові світи з унікальними характеристиками.

Процедурна генерація в Minecraft охоплює кілька рівнів, від глобальної структури світу (біоми, гори, океани) до дрібних деталей, таких як розподіл дерев, печер, руд та інших ресурсів.

Механізм генерації часто будується на основі накладення декількох шарів процедурного шуму, кожен з яких відповідає за певний аспект світу. Наприклад, один шар може відповідати за висоту ландшафту, інший – за розміщення водойм, а третій – за розподіл дерев і руд. Таким чином, кожен елемент світу породжується незалежно, але в узгодженні з іншими елементами, що забезпечує складну та

логічну структуру світу.

Підземний світ у Minecraft також генерується процедурно. Це стосується як системи печер, так і розподілу руд і ресурсів. Печери створюються за допомогою клітинного шуму, що дозволяє утворювати розгалуження та випадкові тунелі. Ресурси (наприклад, вугілля, залізо, алмази) випадково розташовані по карті, що спонукає гравця досліджувати світ і знаходити нові родовища руди.

Однією з ключових переваг процедурної генерації в Minecraft є її вплив на відтворюваність. З кожним новим світом, створеним випадковим чином, гравці можуть починати гру знову і знову, щоразу отримуючи унікальний досвід. Це збільшує довговічність гри та підтримує гравців протягом тривалого часу. Додатковим фактором, який покращує відтворення, є можливість впливати на процес генерації, вводячи власне початкове значення, що дає гравцям інструмент для створення світу із заздалегідь визначеними параметрами або дослідження світів, створених іншими гравцями.

У жанрі tower defense процедурна генерація ще не набула такого поширення, як у рогаликах або open-world іграх, але є кілька прикладів успішного використання цього підходу.

Одним з найвідоміших проектів є гра Dungeon Warfare, яка частково використовує процедурну генерацію для створення рівнів. Хоча базова структура рівня гри заздалегідь розроблена, певні елементи, наприклад розміщення пасток або ворогів, можуть відрізнятися в кожній новій грі.

Рис1.4 Рівень з декількома шляхами у Dungeon Warfare

Процедурна генерація в Dungeon Warfare в основному зосереджена на створенні варіативності поведінки ворогів, їхніх маршрутів і розподілу пасток і подій на рівнях. Хоча основна структура карти може залишатися фіксованою, елементи, які впливають на складність проходження, змінюються випадковим чином. Це дає гравцям відчуття, що навіть у знайомому рівні завжди є елемент новизни, який вимагає адаптації.

Один із головних аспектів процедурної генерації пов’язаний зі створенням ворожих хвиль. На відміну від статичних сценаріїв, де вороги з’являються в заздалегідь визначеному порядку, у грі використовується випадкова генерація складу та порядку хвиль. Процес починається з визначення загальної складності рівня, після чого гра генерує послідовності ворогів на основі поточного прогресу гравця та доступних пасток.

Іншим важливим аспектом є створення випадкових маршрутів для ворогів. Хоча карти мають фіксовані точки входу та виходу, гра може змінювати шляхи, якими йдуть вороги. Ці зміни включають створення альтернативних маршрутів або перешкод, які змушують ворогів йти довшими чи небезпечнішими шляхами, а також створення нових зон для проходу ворогів, що збільшує складність і вимагає від гравця більш продуманого розміщення пасток. Процедурна генерація шляху забезпечує динаміку на рівнях, оскільки гравець не може заздалегідь передбачити, яким шляхом підуть вороги, що спонукає до імпровізації та швидкої адаптації.

Щоб досягти високого ступеня різноманітності в ігровому процесі, Dungeon Warfare використовує кілька ключових процедурних алгоритмів і методів генерації, одним з яких є псевдовипадкові числа (PRNG):

Псевдовипадкові числа (PRNG) відіграють важливу роль у визначенні поведінки процедурної генерації. Ці алгоритми дозволяють грі випадково вибирати типи ворогів, маршрути та пастки на рівні, гарантуючи дотримання певних правил і обмежень. Наприклад, генерація ворожої хвилі підлягає суворим обмеженням, щоб рівень не був надто складним або надто легким. У той же час використання PRNG забезпечує достатню різноманітність, щоб кожен ігровий сеанс був унікальним.

У деяких режимах гри, особливо в нескінченному режимі, генерація процедурного рівня використовує алгоритми графів для створення випадкових шляхів і маршрутів ворогів. Ці алгоритми допомагають грі створювати нові зв’язки між існуючими елементами карти, що призводить до нових ігрових ситуацій. Кожен новий маршрут побудований таким чином, щоб зберегти баланс складності та надати гравцеві нові тактичні варіанти.

Іншим прикладом є Revenge of the Titans, яка також використовує процедурну генерацію для створення місцевості та шляхів ворога. Це дозволяє різноманітний ігровий процес і підтримує інтерес гравців, оскільки кожен новий рівень вимагає іншого підходу до створення захисту.

У багатьох сучасних стратегіях і іграх для створення бази також використовуються методи процедурної генерації для створення карт і рельєфу. Наприклад, Civilization VI використовує складний алгоритм для процедурного створення карт, причому кожна гра починається на унікальному континенті з випадковим розташуванням ресурсів, річок, гір і лісів. Це дозволяє гравцям щоразу адаптувати свої стратегії до нових умов середовища.

Основною метою процедурної генерації є створення реалістичних і різноманітних карт, які слугуватимуть основою для глобальних стратегічних рішень. Вона охоплює кілька ключових аспектів ігрового світу: місцевість, ресурси, розташування міст-держав, варварів, а також природні та штучні перешкоди, такі як гірські хребти, пустелі, річки та океани. Ці елементи створюють унікальні умови для кожної нової гри, впливаючи на розвиток цивілізацій і визначаючи стратегічні варіанти гравців.

Процедурно згенерований світ є ключовим елементом, який впливає на всі аспекти ігрового процесу, від розміщення ресурсів і міст до дипломатичних відносин і військових дій.

Процедурна генерація в Civilization VI спирається на кілька складних алгоритмів, щоб забезпечити створення різноманітних і збалансованих ігрових карт.

Основним інструментом гри для процедурної генерації рельєфу є алгоритм Perlin Noise. Цей метод дозволяє плавно переходити між різними типами місцевості та різноманітними формами континентів. Perlin Noise працює з псевдовипадковими числами, що забезпечує унікальність кожного світу, але в той же час зберігає шаблони, які роблять карти реалістичними.

Діаграми Вороного допомагають розділити карту на логічно пов'язані зони. Цей метод дозволяє визначати зони впливу різних цивілізацій, а також розподіляти важливі об’єкти, наприклад ресурси чи міста-держави. Діаграми створюють «комірки», які потім обробляються з урахуванням локальних особливостей карти та стратегічних пріоритетів.

Подібний підхід можна побачити в таких іграх, як Age of Wonders: Planetfall, де карти процедурно генеруються з урахуванням різних параметрів, таких як клімат, біоми та розташування стратегічних об’єктів. Це робить кожну ігрову сесію унікальною та додає нові виклики для планування та керування ресурсами.
Шумові функції, такі як шум Перліна, широко використовуються в сучасних іграх для створення природних ландшафтів, текстур і структур. У грі «No Man's Sky», використання процедурної генерації на основі шуму дозволяє створювати мільйони унікальних планет зі своїми ландшафтами, флорою і фауною. У грі використовується складна система шуму для створення реалістичних ландшафтів, які виглядають природними та органічними, навіть якщо вони були створені автоматично.

Так само гра Terraria використовує шумові функції для створення підземних шахт і печерних систем. Це дозволяє генерувати унікальні структури, які змінюються в кожному новому світі, підтримуючи постійний інтерес гравців до дослідження та пошуку нових ресурсів.

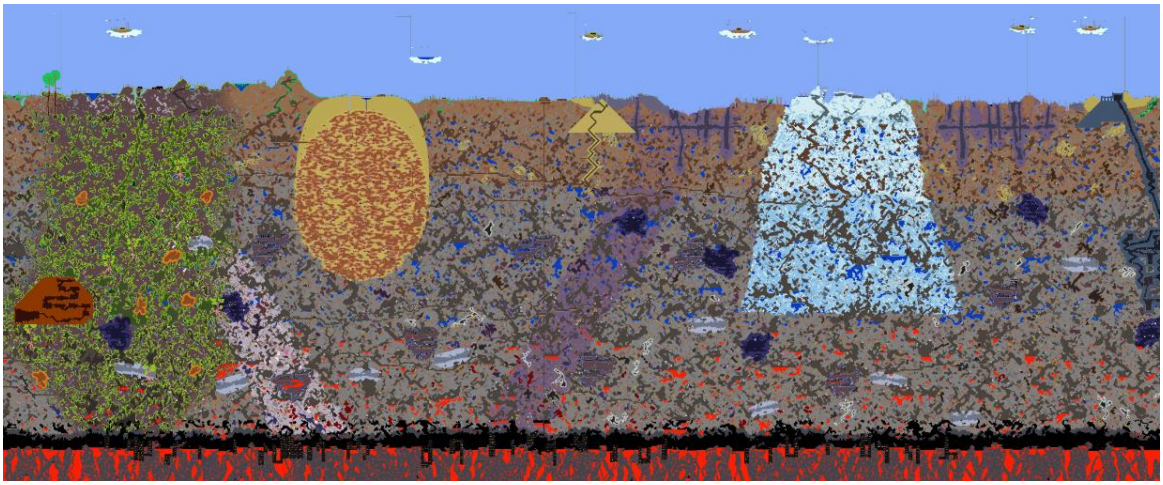


Рис 1.5 Процедурно згенерована мапа у Terraria

Одним з аналогів процедурної генерації є ручне створення світу, що донедавна було основним методом розробки ігрових світів. Ручна робота над кожним елементом гри дозволяє розробникам детально продумати структуру рівнів, оптимізуючи їх для ідеального балансу складності, сценаріїв і естетики. Найчастіше цей підхід використовується у великих проектах, де високі бюджети дозволяють наймати великі команди дизайнерів, художників і програмістів. Однак, незважаючи на певні переваги ручного створення контенту, цей метод має значні недоліки порівняно з процедурною генерацією, особливо в жанрі tower defense.

Створення рівнів вручну займає багато часу, оскільки кожен елемент світу — від ландшафту до розташування об’єктів і ворогів — має бути розроблений і перевірений вручну. Цей процес часто займає тижні або навіть місяці, особливо якщо в грі велика кількість рівнів або об'єктів. Для невеликих студій з обмеженими ресурсами це може стати серйозною перешкодою, оскільки для розробки якісного контенту потрібен не тільки час, але й висококваліфіковані спеціалісти, що збільшує витрати на проект.

Крім ручного та процедурного створення рівнів є також модульне створення рівнів:

Модульна генерація базується на створенні великої кількості готових елементів або «модулів», які можна комбінувати для створення рівнів або інших аспектів гри. Ці модулі розробляються вручну, але потім збираються випадковим

чином, створюючи ілюзію варіативності.

Модульна генерація починається зі створення набору «модулів» — заздалегідь підготовлених блоків або сегментів ігрового світу. Ці модулі можуть включати в себе різні типи структур: кімнати, коридори, частини будівель, елементи ландшафту і т. д. Залежно від гри, кожен модуль може представляти як невеликий фрагмент середовища (наприклад, кімнату в підземеллі), так і більший структур (наприклад, ділянки міста чи території).

Рис 1.6 Створення унікальних будинків з одного модульного набору

Основна ідея полягає в комбінуванні та з’єднуванні різними способами. Алгоритми визначають, як і в якому порядку будуть розташовані модулі, створюючи унікальні карти або структури, які зберігають певну логіку та внутрішню цілісність. Наприклад, у модульній грі про створення підземель алгоритм може випадковим чином з’єднувати заздалегідь спроектовані кімнати, коридори та зали, створюючи таким чином цілий рівень, але уникаючи таких проблем, як глухий кут або надмірна лінійність.

Однією з ключових переваг модульної генерації є те, що розробники можуть створювати та тестувати кожен модуль вручну, гарантуючи, що він відповідає високим стандартам якості. На відміну від повної процедурної генерації, де складні алгоритми можуть генерувати невдалі або незбалансовані рівні, у модульній

генерації кожна частина вже перевірена та налагоджена.

Одним із успішних прикладів використання модульної генерації буде гра Spelunky:

«Spelunky» — одна з найвідоміших ігор, яка використовує модульну генерацію. Тут рівні формуються з невеликих модулів: кімнат, коридорів, пасток і платформ. Потім ці елементи випадковим чином поєднуються, щоб сформувати унікальні карти на кожному етапі гри. Кожен рівень у грі представлений у вигляді сітки, де кожна клітинка сітки є окремим модулем, який можна заповнити певним вмістом, таким як перешкоди, скрині зі скарбами або вороги.

Алгоритм генерації використовує попередньо розроблені шаблони для модулів, щоб гарантувати, що рівні завжди залишаються структурованими та логічно пов’язаними. Ці шаблони, незважаючи на їх випадкове розташування, дотримуються загальних правил: наприклад, гравець повинен завжди мати можливість дістатися до виходу, а вороги та пастки розміщуються таким чином, щоб створити виклик, не унеможливлюючи його виконання. Цей підхід гарантує, що кожен рівень залишається придатним для гри, зберігаючи при цьому баланс складності та різноманітності.

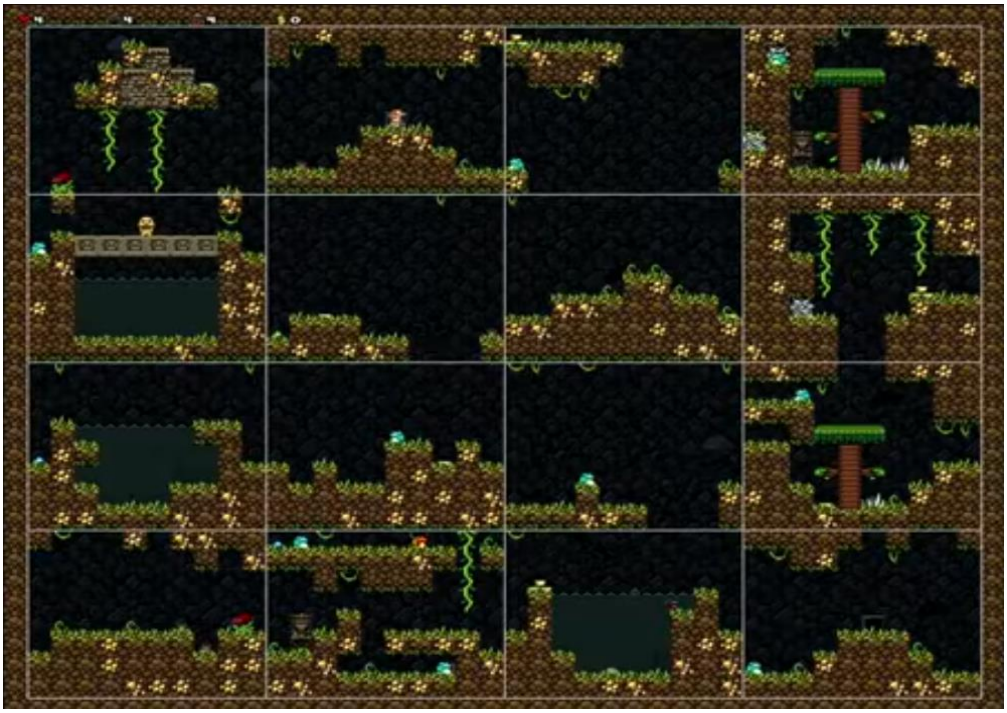


Рис 1.7 Розділення рівню на 16 клітин для модульної генерації

Також Spelunky використовує алгоритм випадкового вибору модулів для створення рівнів. Щоразу, коли гра генерує рівень, вона вибирає випадковий набір модулів із попередньо створеного пулу. Ці модулі можуть містити різні комбінації перешкод, ворогів і об'єктів. Однак випадковість у цій системі контролюється, щоб рівні були збалансованими та не включали занадто багато складних перешкод або ворогів на початку гри.

1.3. Огляд методів та їх переваги

У процесі розробки обраного методу процедурної генерації, важливо спиратися на сучасні методи та підходи, які забезпечують варіативність високого рівня, їх адаптацію до ігрового процесу та оптимальну складність для різних категорій гравців. Процедурна генерація ігрових об'єктів є однією з ключових технологій, яка значно покращує відтворюваність і мінімізує витрати на розробку контенту вручну. Далі будуть розглянуті основні методи, за допомогою яких можна розв'язувати задачі генерації ігрових об'єктів та світу, а також обґрунтування вибору найбільш прийнятних методів для реалізації поставленого завдання.

Одним із ключових підходів у процедурній генерації є використання шумових функцій. Шум дозволяє генерувати псевдовипадкові структури, які в той же час мають певний ступінь плавності та логічності. Шумові функції засновані на математичних алгоритмах, які генерують випадкові значення з урахуванням заданих параметрів, що дозволяє створювати як прості, так і складні структури ігрових світів. Найпоширенішими типами шуму в ігровій індустрії є шум Перліна, шум Уорлі та фрактальний шум.

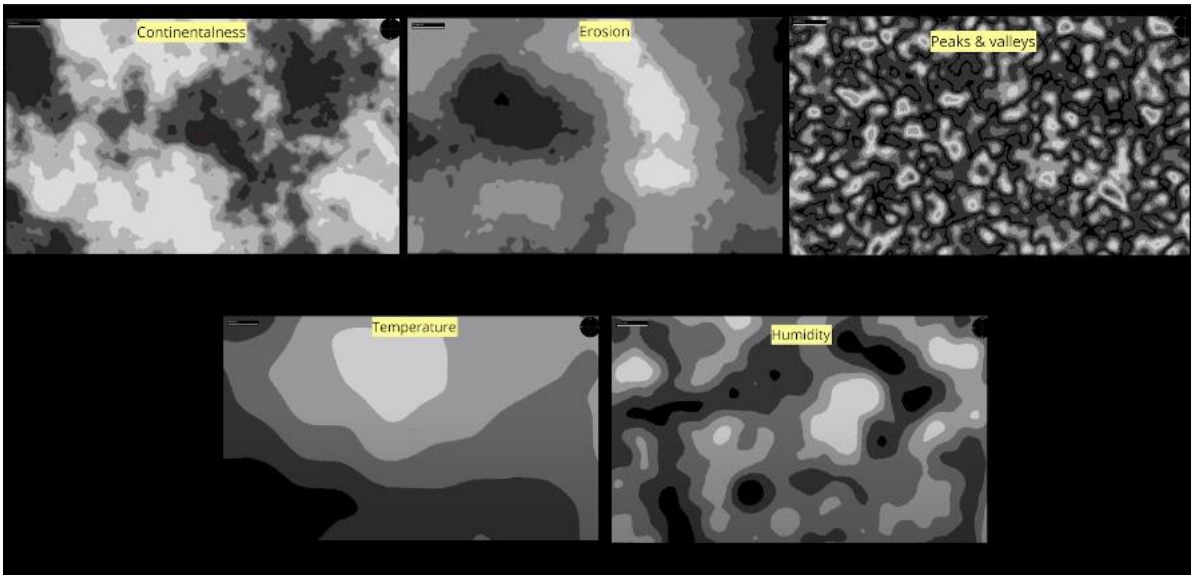


Рис 1.8 Набір різних карт шумів

1. Шум Перліна (Perline Noise) був розроблений Кеном Перліном у 1983 році і є одним із найпопулярніших методів генерації псевдовипадкових даних. Його ключовою особливістю є те, що він створює плавні переходи між різними значеннями, що робить його ідеальним інструментом для створення природних ландшафтів, таких як гори, рівнини та інші географічні форми. На відміну від класичних випадкових функцій, шум Перліна дозволяє уникнути різких стрибків значень, забезпечуючи більш природні та реалістичні структури. У розробці методу, шум Перліна буде використовуватися для створення ландшафтів і розподілу ключових ігрових об'єктів. Це дозволить досягти органічних переходів між різними зонами рівня та забезпечити різноманітність генерованих карт.

З плюсів можна виділити генерування плавних та органічних структур та легке налаштування амплітуд та частот.

2. Шум Уорлі (Worley Noise), також відомий як клітинний шум, використовується для створення структур із чіткими межами, наприклад кластерів об’єктів або зон. На відміну від шуму Перліна, який створює плавні переходи, шум Уорлі більше підходить для генерації об’єктів із чітким розділенням, таких як розподілені групи ворогів або зони з різними характеристиками. Його використання підійде для створення сегментованих зон на карті, де будуть розташовані війська або об’єкти противника. Це дозволить створювати різні бойові

сценарії та стратегічно важливі напрямки.

Можливість створення кластерів дозволяє використовувати цей метод для групування об’єктів, таких як бази противника, будівлі або оборонні споруди.

3. Фрактальний шум — це комбінація кількох шумових функцій, які накладаються одна на одну з різними частотами й амплітудами. Цей підхід дозволяє створювати складні та деталізовані структури, які можна використовувати для створення складних ігрових світів із багаторівневим ландшафтом. Фрактальний шум буде використовуватися для створення більш складних рівнів з різними географічними особливостями, такими як пагорби, річки та інші елементи ландшафту. Це забезпечить стратегічну складність і різноманітність карт.

Головною перевагою фрактального шуму є можливість створення деталізованих ландшафтів та об'єктів за рахунок комбінування кількох шумів.

4. Алгоритми найкоротшого шляху, такі як алгоритм A* і алгоритм Дейкстри, широко використовуються в ігровій індустрії для пошуку оптимальних маршрутів між точками на карті. Ці методи дозволяють ефективно розраховувати шляхи ворога, уникаючи перешкод і враховуючи рельєф карти. Алгоритми пошуку шляхів використовуватимуться для створення маршрутів, якими вороги слідуватимуть протягом усього рівня, дозволяючи ворогам знаходити оптимальні шляхи до точок атаки, враховуючи захист гравця та перешкоди на карті.

5. Binary Space Partitioning (BSP) — метод поділу ігрового простору на зони або секції для спрощення структуризації рівнів. Цей метод дозволяє розділити карти на логічні області, які потім заповнюються об’єктами або з’єднуються шляхами.

6. Важливою частиною розробки є також використання адаптивних методів генерації, які дозволять динамічно регулювати складність рівнів і розташування об'єктів в залежності від стилю гри і прогресу гравця. Особливо це актуально для жанру Tower Defense, де важливий баланс між складністю та інтересом для гравця.

7. Методи динамічного налаштування складності (DDA) дозволяють регулювати параметри рівня гри на основі навичок і прогресу гравця. Ці методи забезпечують більш персоналізований досвід для кожного гравця, допомагаючи підтримувати їхню активність протягом усієї гри.

Виходячи з перерахованих вище методів, найбільш відповідними для вирішення завдання будуть:

1. Шуми Перліна та інші шумові функції, так як вони використовуються для створення природних, органічних топологій у процедурах генерації рівнів. Цей метод дозволяє генерувати плавні безперервні зміни висоти та інших параметрів карти, створюючи реалістичні ландшафти, такі як пагорби, долини чи ліси. Зокрема, цей метод підходить для створення рельєфу карти та визначення різних типів поверхонь, таких як водойми чи пагорби.

2. Модульна генерація (Tiled-based метод) - модульний метод генерації, який використовує елементи у вигляді заздалегідь підготовлених модулів або «тайлів», є ефективним інструментом процедурного створення карти. У цьому випадку карта рівнів представлена у вигляді сітки, де кожен тайл (комірка) може бути заповнений певним типом поверхні або об'єкта. Наприклад, тайл може представляти трав'яне поле, кам'яну поверхню або водойму. Генерація тайлів дозволяє легко керувати процесом розміщення об'єктів і уникати непрохідних або нелогічних ділянок карти. Цей метод добре підходить для завдань, пов'язаних із розміщенням об'єктів оточення, так як об'єкти можуть бути розміщенні на основі типу місцевості, що генерується кожною клітиною сітки.

3. Алгоритми розподілу об'єктів оточення (Object Placement Algorithms): Після створення базової карти з рельєфом необхідно згенерувати об'єкти середовища, такі як дерева, каміння, вежі та інші елементи декору. Для цього використовується алгоритм випадкового або керованого розподілу об'єктів залежно від типу поверхні. Наприклад, для процедурного розміщення дерев можна використовувати алгоритм випадкового розподілу з фільтрацією, який враховує особливості рельєфу - наприклад, дерева можуть рости тільки на траві, а каміння – на скелястій поверхні.

Отже, з обраних методів можна виділити основні переваги:

Гнучкість і варіативність: такі функції шуму, як шум Перліна, дозволяють гнучко керувати створенням рельєфу карти, створюючи гладкі та природні ландшафти. Це робить кожен рівень унікальним, але зручним і збалансованим.

Спрощення розробки. Модульна генерація за допомогою плиток і біомів забезпечує зручну систему для створення карт, куди можна додавати елементи навколишнього середовища на основі попередньо визначених правил. Це спрощує процес розробки рівня, оскільки творцю гри не потрібно вручну розміщувати кожен об’єкт.

Логічний розподіл об'єктів. Алгоритми розподілу об’єктів середовища забезпечують логічне та природне розміщення різних елементів на карті, створюючи реалістичні сценарії ігрового світу. Це важливо для забезпечення візуальної цілісності та атмосфери карти.

Внаслідок цього, поєднання методів генерації шуму, модульної структури та системи розподілу дозволяє ефективно вирішувати задачу процедурної генерації карт та об’єктів навколишнього середовища, забезпечуючи варіативність, реалістичність та керованість процесів, що є ключовою вимогою даної роботи.

1.4. Постановка наукового завдання дослідження

Виходячи із зазначеної вище актуальності проблеми, дане дослідження спрямоване на вирішення ряду проблем, пов’язаних із створенням унікальних та різноманітних ігрових рівнів. Процедурна генерація контенту є важливим інструментом у сучасній розробці ігор, що дозволяє не тільки скоротити трудовитрати на створення рівнів, але й зберегти високу реіграбельність за рахунок автоматичного створення нових сценаріїв. У контексті жанру Tower Defense це особливо важливо, оскільки кожен новий рівень повинен представляти гравцеві нові тактичні завдання, роблячи ігровий процес непередбачуваним і захоплюючим.

Завданням даного дослідження є розробка методу процедурної генерації рівнів на основі шуму, який дозволить генерувати як ігрові об’єкти, так і структуру карт з урахуванням специфіки жанру Tower Defense. Під час дослідження будуть

вирішуватися такі завдання:

1. Розробити метод процедурної генерації карти на основі використання шуму Перліна та інших алгоритмів генерації шуму. Цей метод має дозволити створювати різноманітні топології рівнів, включаючи нерівності поверхні, зони розміщення об’єктів навколишнього середовища та зони, які визначатимуть унікальний візуальний стиль кожної карти.
2. Проектування системи автоматичного розміщення об'єктів довкілля на сформованих картах. Необхідно розробити алгоритм, який буде враховувати тип місцевості та особливості ландшафту для правильного розподілу таких елементів, як дерева, скелі, водойми та інші об'єкти, що впливають на візуальне сприйняття ігрового світу.
3. Створення бібліотеки об'єктів середовища, які будуть використовуватися при генерації карти. Кожен об'єкт середовища повинен бути підготовлений заздалегідь, а алгоритм генерації повинен відповідати за їх випадкове і логічне розміщення в залежності від особливостей ландшафту карти.

Метою дослідження є створення методу генерації ігрових карт і середовищ, який дозволить автоматизувати процес створення контенту для ігор Tower Defense. Результати роботи нададуть розробникам інструмент для генерації різноманітних карт з унікальними об’єктами навколишнього середовища, що дозволить скоротити час і ресурси, необхідні для створення ігрового світу вручну.

Цей метод буде корисний як інді-розробникам, так і студіям, що створюють ігри з великою кількістю рівнів, які вимагають візуальної варіативності та привабливості. Кінцевий продукт дослідження може бути використаний для автоматизації процесу розробки карт і створення процедурно згенерованих світів, забезпечуючи високий ступінь унікальності для кожного рівня.

2. МАТЕМАТИЧНІ МЕТОДИ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ІГРОВИХ КАРТ

2.1. Математичні моделі шумів та їх застосування

Процедурна генерація - один із ключових напрямків у сучасній розробці ігрових світів. Завдяки їй ігрові карти набувають унікальності, забезпечуючи повторюваність ігрового процесу без прямого втручання розробника. Одним із найпоширеніших інструментів реалізації процедурної генерації є функції шуму, які дозволяють створювати карти з плавними переходами висот, текстур або класифікаційних зон (наприклад, води, рівнини, гори). Ці функції стали основою багатьох сучасних алгоритмів генерації, особливо у жанрі Tower Defense, де різноманітність карт та його складність безпосередньо впливають якість ігрового процесу.

Процедурна генерація базується на створенні контенту за допомогою алгоритмів та математичних моделей, а не ручного моделювання кожного елемента. Такий підхід дозволяє скоротити час розробки, автоматизуючи процес створення ігрових елементів, збільшити різноманітність ігрових локацій, створюючи карти, які не повторюються та оптимізувати використання пам’яті шляхом збереження параметрів, а не готових текстур або моделей.

Одним з базових компонентів такої генерації є шумові функції, які використовуються для створення основи ігрового світу.

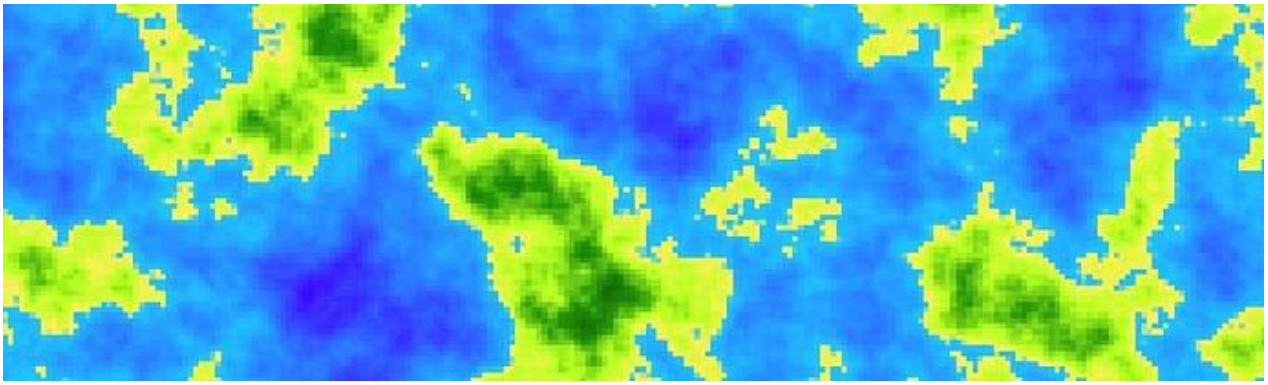


Рис 2.1 Карта згенерована шумом

Шум у контексті генерації карт – це математична функція, яка повертає значення на основі координат точки у просторі. Наприклад, шум Перліна, одна з найпопулярніших функцій, генерує псевдовипадкові значення, що виглядають як плавний перехід між значеннями в сусідніх точках.

Псевдовипадкові значення є ключовим елементом процедурної генерації з використанням шумових функцій. Вони створюють ілюзію випадковості, але ґрунтуються на суворих математичних алгоритмах. Це дозволяє отримувати результати, що відтворюються, зберігаючи при цьому контроль над процесом генерації.

На відміну від істинно випадкових значень, які виникають через неконтрольовані процеси (таких як тепловий шум або радіоактивний розпад), псевдовипадкові значення генеруються алгоритмами і є детермінованими. Вони виглядають випадковими для зовнішнього спостерігача, відтворювані, тобто той самий набір вхідних параметрів (seed) завжди дає той самий результат та контрольовані, завдяки чому розробник може налаштовувати параметри, щоб змінювати характер псевдовипадковості.

Псевдовипадкові значення є основою створення плавних переходів у шумі. Наприклад, в шумі Перліна, для кожної точки сітки обчислюється псевдовипадковий градієнт, який визначає напрямок і силу впливу цієї точки на результуюче значення шуму.

Загалом, можна виділити три етапи їх використання:

1. Генерація псевдовипадкових чисел для вузлів сітки: На кожній точці сітки простору зберігається число або вектор, отриманий за допомогою псевдовипадкової функції.

2. Інтерполяція: Перехід між вузлами забезпечується за рахунок математичних операцій, таких як згладжувальні функції. Завдяки цьому результат виглядає природно.

3. Повторюваність: Використання фіксованого seed дозволяє отримувати однакові результати для тих самих вхідних параметрів.

У шумових функціях використовуються різні алгоритми для отримання

псевдовипадкових значень, такі як:

1. Лінійний конгруентний метод (LCG): простий і швидкий алгоритм, який створює псевдовипадкові числа шляхом модульних операцій.

2. Функції гешування: використовуються для генерації повторюваних псевдовипадкових значень на основі координат точки. Наприклад, гешування дозволяє для однакових координат завжди отримувати однакові результати.

3. Алгоритми перетворення: наприклад, Box-Muller для перетворення рівномірно розподілених випадкових чисел у нормально розподілені.

Жанр Tower Defense вимагає створення карт, які не лише візуально привабливі, а й функціональні. Гравець повинен мати чітке уявлення про структуру карти, щоби будувати стратегії. Шум використовується для створення базового рельєфу, розташування регіонів або структур, а за допомогою порогових значень шуму, можна виділяти області для певних типів ландшафтів (наприклад, місця для розташування дерев, водних ресурсів, або скель) та за допомогою випадковості кожна нова карта виглядає унікальною завдяки псевдовипадковості значень.

Однією з перших функцій, що набули широкого поширення, став шум Перліна, запропонований Кеном Перліном в 1983 для створення текстур у фільмі «Трон». Ця функція швидко стала стандартом у комп'ютерній графіці завдяки простоті реалізації та можливості генерувати плавні текстури. У подальшому розвиток шумових функцій привів до створення багатьох інших карт шуму.

Роботу шуму можна поділити на три кроки:

1. Простір розбивається на сітку, де кожна точка має фіксовані координати.

2. Для кожної точки сітки генерується псевдовипадкове значення або градієнт.

3. Коли точка простору потрапляє між вузлами сітки, результат розраховується за допомогою інтерполяції, яка забезпечує плавний перехід між значеннями.

Наприклад, у шумі Перліна використовується спеціальна fade-функція для згладжування переходів між точками:

$$fade(t) = 6t^5 - 15t^4 + 10t^3, \quad (2.1)$$

де t – параметр інтерполяції, який змінюється від 0 до 1,

Степені t^5, t^4, t^3 — використовуються для створення плавних кривих, що починаються і закінчуються на нульовому градієнті.

Fade-функція (або згладжування) є ключовим елементом в алгоритмах шуму, таких як шум Перлина. Її основне завдання – забезпечити плавний перехід між значеннями шуму в різних точках сітки. Цей плавний перехід створює природну текстуру або рельєф без різких перепадів.

Сама по собі, fade-функція це математична функція, яка згладжує значення між вузлами сітки. Її основна мета полягає у зменшенні артефактів, щоб переходи між сусідніми точками були плавними та усували різкі зміни у значеннях. Також результати виглядають більш реалістичними для ока, імітуючи явища природи.

У шумі Перліна fade-функція застосовується до параметрів інтерполяції (координат), щоб надати більш плавний і природний вигляд результату.

Ця функція обирається через її математичні властивості:

1. Вона має нульову похідну на краях інтервалу ($t = 0$ і $t = 1$), що запобігає появі різких змін

2. Забезпечує плавне зростання і зменшення значення, мінімізуючи артефакти.

Принцип праці цієї функції забезпечується декількома факторами:

1. Координати точки: Після визначення позиції точки $P(x,y)$ у просторі шуму, обчислюється відстань до найближчих вузлів сітки (точок решітки).

2. Згладжування: Значення t, яке є відстанню від точки до вузла, передається у fade-функцію.

3. Інтерполяція: Згладжене значення $f(t)$ використовується для зваженої лінійної або градієнтної інтерполяції між значеннями шуму у вузлах сітки.

Наприклад Якщо точка $P(x,y)$ знаходиться між двома вузлами A іB, то $f(t)$ визначає, як близькість до одного вузла впливає на загальне значення шуму в цій

точці.

Без fade-функції результати шуму виглядали б надто штучно через різкі переходи між значеннями. Хоча fade-функція є стандартом для шуму Перліна, для цього завдання можна використовувати інші функції, наприклад лінійну функцію, без згладжування, але вона даватиме різкі зміни.

Іншим варіантом може бути кубічна функція:

$$f(t) = 3t^2 - 2t^3. \tag{2.2}$$

Вона простіша, і менш “гладка”.

Розглядаючи сам шум Перліна, можна виділити його ґрунтування на регулярній сітці точок у просторі. Ця сітка складається з вузлів (граток), у кожному з яких визначений псевдовипадковий градієнтний вектор.

У 2D сітка визначає двовимірну площину, на якій формується шум, а у 3D сітка додає ще один вимір, дозволяючи створювати тривимірні текстури, наприклад, для об’ємних моделей.

У кожному вузлі сітки обчислюється псевдовипадковий вектор, спрямований у випадковому напрямку і визначає зміну значень шуму навколо цього вузла.

Математичні формули шуму Перліна виглядають наступним чином:

$$i = floor(x), j = floor(y) \tag{2.3}$$

$$g_i j, g_i + Ij, g_i j, g_i j + I, g_i + I, j + 1 \tag{2.4}$$

$$d_{i,j} = (x - i, y - j) \tag{2.5}$$

$$s = g_{i,j}, t = g_{i+Ij}, u = g_{i,j} \cdot d_{i,j} + 1, v = g_{i+1,j+1} \cdot d_{i+1,j+1} \tag{2.6}$$

$$l_1 = lerp(s, t, f(x - i)) \tag{2.7}$$

$$l_2 = lerp(u, y, f(x - i)) \tag{2.8}$$

$$noise(x, y) = lerp(l_1, l_2, f(y - j)) \quad (2.9)$$

Де перша формула відповідає за розбиття простору на сітку, кожна точка $P(x,y)$ простору відноситься до комірки сітки. Визначаються координати найближчих вузлів сітки.

Формула (2.3) визначає градієнтні вектори.

Для кожного з чотирьох вузлів у 2D-сітці (нижній лівий, нижній правий, верхній лівий, верхній правий) задаються псевдовипадкові градієнтні вектори.

Формули (2.4), (2.5) обчислює скалярний добуток, за допомогою якого визначаються вектори від кожного вузла до точки $P(x,y)$.

Формули (2.6), (2.7), (2.8) відповідають за інтерполяцію, результати скалярного добутку якого згладжуються за допомогою fade-функції. Спочатку виконується інтерполяція значень уздовж осі x, а потім результати інтерполюються вздовж осі y, де:

$$lerp(a, b, t) = a + t(b - a) \quad (2.10)$$

функція лінійної інтерполяції.

f(t) — fade-функція для згладжування.

Візуалізація шуму є важливим кроком під час його використання у процедурній генерації. Завдяки графічному уявленню можна оцінити якість шуму, протестувати його параметри і зрозуміти, як він впливає на кінцевий результат, наприклад, на карту місцевості або текстуру.

Шум Перліна та інші функції шуму зазвичай відображаються у вигляді двовимірної карти, де кожна точка (піксель) відповідає значенню шуму в координатах (x,y) . Значення шуму нормалізуються в діапазоні від 0 до 1, що дозволяє їх відобразити у відтінках сірого: чорний колір (0) відповідає за мінімальне значенню шуму, а білий (1) за максимальне значення.

У шуму можна виділити три параметри:

1. Частота: частота визначає масштаб шуму. Зі збільшенням частоти

структури стають меншими, а шум здається більш докладним. І навпаки, зменшення частоти робить структури більш і менш детальними. За низької частоти утворюються великі та плавні переходи між відтінками, а при високій частоті чіткі та дрібні деталі.

2. Амплітуда: амплітуда визначає максимальну різницю між значеннями шуму. Збільшення амплітуди збільшує контраст між темними та світлими областями.

3. Октави: октави використовуються для накладання шуму. Кожна октава додає новий шар шуму з більш високою частотою та нижчою амплітудою. Це створює складнішу текстуру.

Для більшої інформативності значення шуму можуть бути представлені кольоровою картою (градієнтом). Це особливо корисно для процедурної генерації ландшафтів:

Рис 2.2 Кольорова (градієнт) мапа шуму Перліна

На рисунку 2.2 можна виділити декілька кольорів, які б могли означати різні регіони: Вода, пісок, рівнини та вершини. Кожен регіон має відповідний колір,

синій, жовтий, зелений та помаранчевий.

Такий підхід дозволяє миттєво оцінити розподіл висот і структуру рельєфу.

Шум може бути представленим у динаміці. Наприклад, тривимірний шум у просторі (x,y,z), де z змінюється з часом, дозволяє створювати анімації, що імітують хвилі, рух хмар або зміни текстури з часом.

2.2. Алгоритмічний аналіз процедурної генерації карт

Блок-схема алгоритму процедурної генерації демонструє основні етапи роботи системи, яка створює карту з використанням шумів Перліна. Алгоритм охоплює ініціалізацію параметрів, використання шумових функцій для визначення властивостей регіонів і розміщення об’єктів, а також перевірку валідності створених регіонів.

На початковому етапі виконуються всі дії, необхідні для підготовки до роботи алгоритму. Далі наведені важливі кроки цього етапу:

1. Визначення стартових параметрів. Задаються масштаб карти, розміри сітки, кількість регіонів і деталізація шуму. Ці параметри задають рамки для роботи алгоритму і впливають на кінцевий вигляд карти.

2. Створення псевдовипадкових значень. Використання початкового seed забезпечує відтворюваність карти, що є важливою вимогою для стабільної роботи системи.

3. Задання конфігурацій для шуму Перліна. Встановлюються початкові характеристики, такі як частота, амплітуда, масштаб і інші властивості функції шуму. Від цих параметрів залежить, наскільки природно виглядатимуть переходи між різними регіонами.

Підготовчий етап є базою для наступних дій, через правильно задані параметри забезпечується передбачувана і коректна робота всього алгоритму.

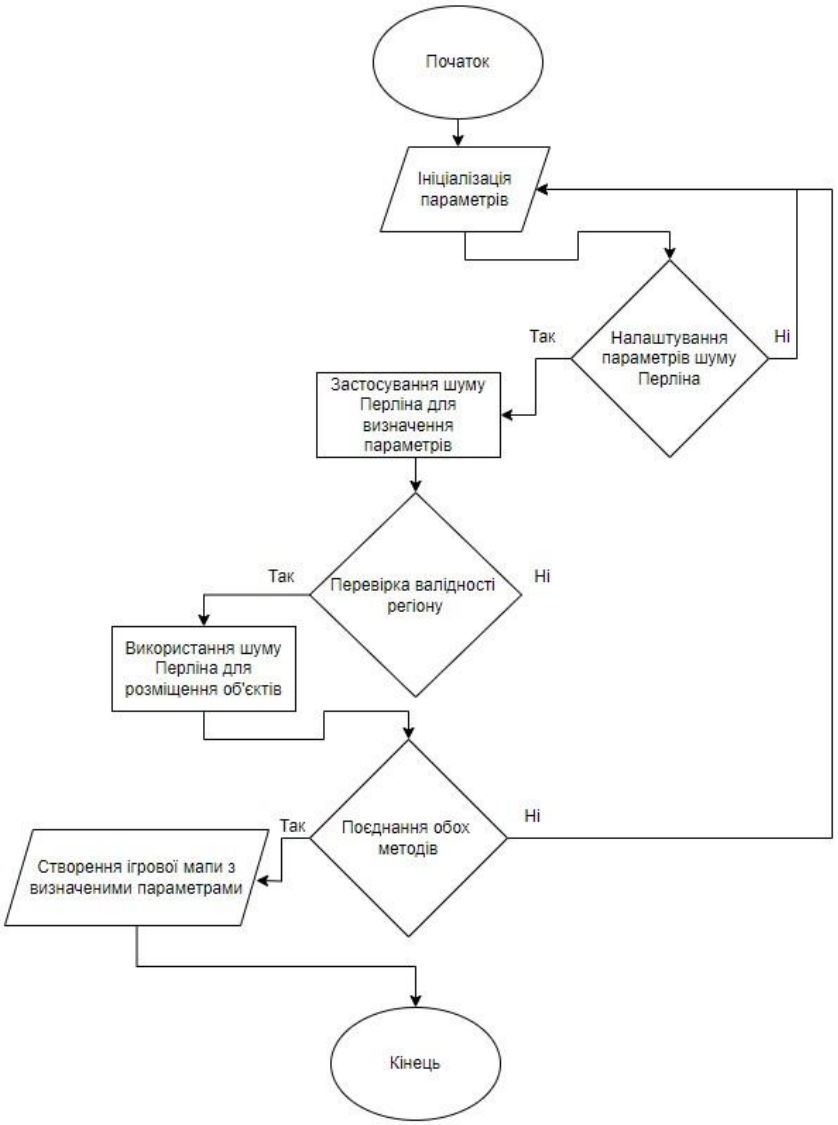


Рис 2.3 Блок-схема методу

Наступний етап — генерація карти на основі шуму Перліна. Тут відбувається створення базових параметрів карти:

1. Спочатку йде розрахунок значень шуму для кожної точки. Кожна координата карти (x, y) обробляється за допомогою шуму Перліна, що генерує плавний перехід між висотами або іншими властивостями. Це дозволяє уникнути різких змін та розривів, властивих звичайному випадковому розподілу.

2. Корекція шуму. У разі необхідності алгоритм виконує додаткові обчислення, наприклад, змінює масштаб або частоту, щоб адаптувати результати до конкретного завдання.

3. Регіональна сегментація. На основі значень шуму визначаються області карти, які відповідають певним типам регіонів, таким як вода, рівнини або вершини.

Після розташування об’єктів на мапі виконується перевірка відповідності кожного об’єкта до його регіону. Цей процес можна поділити на порівняння об’єкта з регіоном та пропуск об’єктів із невідповідним регіоном.

Для кожного об’єкта визначається, чи відповідає його тип властивостям регіону. Наприклад, якщо дерево потрапило в регіон вершин, об’єкт вважається недійсним і алгоритм переходить до наступного об’єкта.

Такий підхід дозволяє уникнути нелогічного розміщення елементів карти, створюючи правдоподібніший світ. Крім того, перевірка регіону допомагає адаптувати карту до різних потреб гри.

На останньому етапі алгоритм передбачає фінальне формування карти та оптимізацію:

1. Після перевірки та розміщення всіх об'єктів результати генерації шуму поєднуються з іншими параметрами, такими як висота, текстура чи фізичні властивості.

2. Далі карта оптимізується для подальшого використання у ігровому середовищі. Це включає конвертацію отриманих даних у формат, придатний для ігрового двигуна.

Блок-схема алгоритму наочно відображає всі етапи роботи, дозволяючи легко виявити можливі недоліки чи оптимізації. Кожен етап чітко розписаний, що сприяє кращому розумінню алгоритму як розробниками, і майбутніми користувачами. Такий структурований підхід підвищує ефективність генерації та полегшує подальший розвиток проекту.

Далі йде створення регіонів, математичний підхід до класифікації регіонів ґрунтується на використанні шуму Перліна, що забезпечує плавні переходи між зонами та дозволяє формувати природні ландшафти. Поділ карти на регіони відбувається шляхом порівняння значень шуму з певними граничними значеннями, що забезпечує простий та ефективний алгоритм створення різноманітного ігрового

світу. Цей підхід також дозволяє інтегрувати додаткові об'єкти в залежності від типу регіону, що додає реалістичності та логіки у створення карти.

Першим етапом алгоритму є визначення типу регіону для кожної точки (x,y) на основі значення шуму $N(x,y)$, яке нормалізоване в діапазоні [0, 1]. Формула класифікації описується такою функцією:

$$R(x,y) = \begin{cases} Water, & \text{якщо } N(x,y) \leq T_{water} \\ Sand, & \text{якщо } T_{water} < N(x,y) \leq T_{sand} \\ Grass, & \text{якщо } T_{sand} < N(x,y) \leq T_{grass} \\ Mountain, & \text{якщо } N(x.y) > T_{grass} \end{cases}, \quad (2.11)$$

де $R(x,y)$ – тип регіону у точці (x,y)

T_{water} , T_{sand} , T_{grass} - порогові значення шуму для різних типів регіонів

Це дозволяє точно розділити карту на логічні зони. Наприклад:

Регіони з низьким значенням шуму ($N(x,y) \leq T_{water}$) визначаються як вода ($Water$). Проміжні значення шуму відповідають піску ($Sand$) або траві ($Grass$), та високі значення шуму, що перевищують T_{grass}, асоціюються з горами ($Mountain$).

Кожен тип регіону може мати унікальні властивості. Наприклад, регіони (Water) можуть бути недоступними для розміщення об'єктів, в той час як регіони (Grass) або (Mountain) можуть відповідати певним типам ресурсів.

Після визначення типу регіону у кожній точці карти виконується перевірка валідності розміщення об'єктів. Це важливий етап, оскільки він забезпечує логічність і реалістичність ігрового світу. Розміщення об’єкта описується функцією $O(x,y)$, яка визначає тип об’єкта, що може бути розміщений у точці(x,y):

$$O(x,y)\begin{cases} O_{Tree}, & \text{Якщо } R(x,y) = Grass \\ O_{Rock}, & \text{Якщо } R(x,y) = Mountain. \\ 0, & \text{Інакше} \end{cases} \quad (2.12)$$

Наприклад, у регіоні (Grass) можуть створитися дерева, у горах каміння або

руда, а якщо регіон не відповідає жодній з умов, на цьому місці нічого не створюється.

Остаточне рішення про розміщення об'єкта приймається на основі функції PlaceObject(x,y):

$$PlaceObject(x,y) = \begin{cases} 1, & \text{Якщо } O(x,y) > T \ i \ R(x,y) \in ValidRegions, \\ 0, & i\text{накше} \end{cases}, \quad (2.13)$$

де O(x,y) – тип об'єкту, розташованого у точці (x,y)

ValidRegions — список регіонів, де дозволено розміщення певного типу об’єкта.

T – граничне значення шуму, яке дозволяє фільтрувати об’єкти.

O_{Tree} - об'єкт типу “Дерево”

O_{Rock} - об'єкт типу “Камінь”

Розглянута модель забезпечує наступні переваги:

1. Гнучкість класифікації. Різні порогові значення дозволяють створювати карти з індивідуальними характеристиками. Наприклад, зменшення порога для T_{water} збільшить площу водяних регіонів, що корисно для створення острівних карт.

2. Логічність розташування об’єктів. Всі об’єкти розміщуються відповідно до свого природного середовища. Наприклад, дерева з’являються лише на трав’яних регіонах, а руда — у горах.

3. Простота реалізації. Модель використовує лише базові математичні функції та умови, що полегшує її інтеграцію у будь-яку ігрову систему.

Обраний метод дозволяє створювати реалістичні ігрові світи із плавними переходами між регіонами, забезпечуючи логічність їх використання. Це створює візуально привабливу карту, яка відповідає ігровим вимогам з реіграбельністю та може бути масштабована під різні сценарії.

2.3. Порівняльний аналіз методів генерації.

Оцінка якості алгоритмів генерації ігрових карт є важливим етапом у розробці процедурних методів, оскільки визначає, наскільки добре карта відповідатиме вимогам проекту. Для визначення якості генерації використовується декілька ключових критеріїв, що дозволяють оцінити різні аспекти результату. Основні метрики, що враховуються під час аналізу:

1. Різноманітність карти. Різноманітність вимірюється кількістю та природою регіонів, що формуються на карті. Чим більше унікальних регіонів (наприклад, ліси, гори, водойми), тим цікавіше та реалістичніше виглядає карта. Різноманітність також враховує розподіл об'єктів: чи вони сконцентровані занадто щільно в певних місцях або залишаються рівномірно розподіленими по всій області.

2. Плавність переходів між регіонами. Ця метрика вимірює, наскільки природно виглядають межі між різними типами регіонів. Різкі переходи, наприклад, між лісом і пустелею можуть мати неприродний вигляд, тому плавність досягається за допомогою шумових функцій, таких як шум Перліна. Генерація має створювати поступові зміни висоти, кольорів чи інших характеристик регіонів.

3. Відповідність заданим умовам. Тут перевіряється, наскільки добре згенерована карта відповідає вихідним параметрам та правилам, заданим розробником. Наприклад, якщо зазначено, що водоймища повинні знаходитися внизу карти, алгоритм повинен це враховувати. Відповідність також поширюється на розташування об'єктів у правильних регіонах (наприклад, дерева в лісах, руда на горах) та масштаб: карта має відповідати необхідним пропорціям регіонів.

4. Ступінь контролю та налаштування. Оцінює, наскільки легко розробник може змінювати параметри генерації (пороги, типи шуму і т. д.) створення карт різних типів чи масштабів. Гнучкість алгоритму забезпечує більшу універсальність та можливість адаптації до різних сценаріїв.

5. Час створення. Час, необхідний для створення карти, також є важливим критерієм, особливо для ігор з великими світами або необхідністю створення в

реальному часі. Цей критерій оцінюється у поєднанні з якістю: чи погіршується продуктивність під час створення складніших карт.

6. Уникнення артефактів. Карта не повинна містити небажаних артефактів, таких як ізольовані об'єкти (наприклад, самотні дерева посеред пустелі) або несподівані різкі перепади висот. Цей контроль допомагає уникнути помилок, які можуть знизити візуальну якість та реалістичність карти.

Експериментальна частина дослідження є важливим етапом, спрямованим на перевірку ефективності розробленого алгоритму створення ігрових карт. В рамках експериментів проводилися тести з використанням кількох видів шуму: шуму Перліна, простого градієнтного шуму та шуму OpenSimplex. Кожна функція оцінювалася за такими параметрами, як масштаб, амплітуда, кількість октав та згладжування.

Шум Перліна є одним із найбільш широко використовуваних алгоритмів у процедурній генерації завдяки своїй здатності створювати плавні та природні переходи між регіонами. Для тестування цього шуму використовувалися такі параметри як масштаб, октави та амплітуди.

Октави є ключовим поняттям у процедурній генерації карток, особливо під час роботи з функціями шуму, такими як шум Перліна або OpenSimplex. Вони є інструментом, який дозволяє додавати деталі до згенерованого шуму шляхом накладання декількох шарів шуму з різними параметрами. Ці шари об'єднуються для створення більш складних та реалістичних структур. Кожна наступна октава має зменшену амплітуду та збільшену частоту. Зазвичай амплітуда кожної октави зменшується в геометричній прогресії, наприклад, удвічі, а частота збільшується, що дозволяє створювати більш дрібні деталі. Наприклад, частота може подвоюватися на кожній наступній октаві.

$$N(x,y) = \sum_{i=0}^{n-1} \Big(\mathrm{Noise}\big(2^i\mathrm{x}, 2^i\mathrm{y}\big)\mathrm{Amplitude(i)}\Big), \qquad (2.14)$$

де $N(x,y)$ – значення комбінованого шуму у точці (x,y).

n – кількість октав.

$Noise(2^i x,2^i y)$ - шум, згенерований на i - й октаві, зі збільшеною частотою.

$Amplitude(i)$ – амплітуда i-ї октави, що зменшується пропорційно до i.

На практиці, чим більше октав, тим більш деталізованим стає результат.

Проте, збільшення кількості октав також підвищує обчислювальну складність алгоритму.

Роль октави дозволяє створювати природні текстури, забезпечувати масштабованість та налаштовувати деталізацію. Без них шум виглядає надто рівномірним або "штучним", октави додають випадковість і різноманітність, роблять карту реалістичною як у великому, так і в дрібному масштабі. Також шляхом вибору кількості октав можна змінювати рівень деталізації та відповідність поставленим завданням.

Повертаючись до порівняння типів шуму, було виділено три параметри:

1. Масштаб: Визначає, як часто регіони повторювалися на карті. Збільшення масштабу давало великі, чітко визначені регіони, які підходять для відкритих просторів, таких як водоймища або рівнини. Зменшення масштабу, з іншого боку, давало невеликі докладні області, але ці карти виглядали менш природно.

2. Октави: Більша кількість октав дозволяла досягти більш високого рівня деталізації. При значеннях від 4 до 6 октав карти виглядали збалансованими з плавними переходами. Однак зі збільшенням кількості октав час обчислень значно збільшувалося.

3. Амплітуда: Цей параметр визначав рівень контрастності між регіонами. Вищі значення амплітуди допомагали зробити різницю між зонами чіткішими, але занадто високий контраст порушував гармонію карти.

Розглядаючи простий градієнтний шум, можна виділити його користь для базових експериментів і швидкого створення простих карт. Його перевагою є низька обчислювальна складність, що дозволяє генерувати карти майже миттєво. З основних функцій в нього є швидша генерація, що зробило градієнтний шум

зручним для створення прототипів або тестових карт, але візуальна якість через це стала нижчою. Різкі переходи між регіонами і відсутність плавності робили карти менш придатними для складних ігрових сценаріїв.

Останнім у порівнянні став шум OpenSimplex. Цей шум покращена версія шуму Перліна, що забезпечує більш згладженні результати. Ця функція виявилася особливо корисною для великих карт, де важливо уникати повторення текстур. Основними перевагами будуть:

1. Стабільно високий рівень деталізації без утворення артефактів. Карти виглядали природно, з м'якими переходами між зонами.

2. Створення гармонійних великих регіонів, які чудово підходили для ігрових середовищ, таких як гори, ліса чи водойми.

3. Час генерації був трохи вищим, ніж у шуму Перліна, але значно нижчим, ніж очікувалося, враховуючи якість результатів.

Кожна шумова функція тестувалася з різними наборами параметрів, було досліджено час обчислення, якість візуалізації та відповідність заданим умовам.

Шум Перліна та OpenSimplex вимагали більше часу на обчислення, але забезпечували значно кращу якість результату. Простий градієнтний шум працював швидше, але його результат був менш привабливим. У якості візуалізації OpenSimplex забезпечив найбільш приємну картинку, тоді як градієнтний шум виглядав занадто "штучно". По відповідності найкраще відповідали завданням шум Перліна та OpenSimplex, генеруючи більш реалістичні ігрові карти. Градієнтний шум був менш передбачуваним.

2.3. Вибір оптимального методу

Для обрання оптимального методу генерації необхідно враховувати специфіку даного жанру, а також цілі, поставлені перед процедурною генерацією. Основною механікою таких ігор є стратегічне управління ресурсами, оборонними спорудами та оптимальне розміщення об'єктів на карті з урахуванням наперед визначених шляхів руху супротивника. Тому алгоритм повинен забезпечувати

баланс між варіативністю, функціональністю та відповідністю дизайну карти до потреб ігрового процесу.

Процедурна генерація у жанрі Tower Defense має вирішувати кілька ключових завдань. По-перше, це створення інтерактивного середовища з логічно структурованими регіонами (наприклад, трава, пісок, вода, гори), які забезпечують чітке розмежування зон із різними ігровими функціями. По-друге, важливо забезпечити плавні переходи між регіонами, щоб уникнути різких та нелогічних змін текстур та ландшафту. І, по-третє, алгоритм повинен мати можливість масштабування та налаштування параметрів для створення сценаріїв різної складності.

Шум Перліна є одним із найкращих варіантів для цих завдань завдяки своїй здатності створювати природні та плавні текстури. Використання шуму Перліна у поєднанні з багаторівневою деталізацією шляхом додавання кількох октав дозволяє генерувати карти, які відповідають вимогам жанру Tower Defense. Наприклад, багаторівнева генерація дозволяє додавати деталі до основних регіонів карти: створювати гірські райони з великою кількістю дрібних об'єктів, таких як каміння або скелі, або гладкі луги з розкиданими деревами. Октави шуму додають складність і реалістичність текстурам, роблячи ігрове середовище більш привабливим для гравця.

Крім того, шум OpenSimplex може використовуватись як альтернатива шуму Перліна. Хоча OpenSimplex вимагає більше обчислювальних ресурсів, він забезпечує більш рівномірний розподіл значень і дозволяє уникнути артефактів, які іноді виникають при використанні шуму Перліна, особливо за високої деталізації. Це може бути важливим для ігор, де унікальність кожної карти або її візуальна привабливість є ключовими факторами.

Для досягнення оптимальних результатів необхідно враховувати особливості ігрового процесу. Такий підхід дозволяє розміщувати дерева тільки в регіонах типу Трава, а каміння - в регіонах типу Гори. Відповідність розміщення об'єктів регіонам додає карті логічності та послідовності, що позитивно впливає на ігровий процес.

Таблиця 3.1

Порівняння методів

Параметр\ Характеристика	**Шум Перліна**	**Шум Simplex**	**Шум Уорлі**	**Білінійний (Grid) шум**
Тип алгоритму	Градієнтний шум	Градієнтний шум	Геометричний шум	Лінійна інтерполяція
Вимоги до обчислень	Помірні	Високі	Низькі	Низькі
Плавність переходів	Висока	Висока	Висока	Низька
Різноманітність карт	Середня	Висока	Висока	Низька
Контроль параметрів	Добрий	Добрий	Обмежений	Мінімальний
Ефективність у 2D-генерації	Висока	Висока	Низька	Помірна
Ефективність у 3D-генерації	Середня	Висока	Низька	Низька
Області застосування	Ландшафти, текстури	Ландшафти, текстури,складні карти	Розташування об'єктів	Базова генерація
Переваги	Простота реалізації	Ефективність у багатовимірних просторах	Природне формування кластерів	Простота та швидкість
Недоліки	Артефакти на великих картах	Вищі витрати обчислювальних ресурсів	Нема контролю параметрів	Низька якість

3. МЕТОДИКА РЕАЛІЗАЦІЇ ТА ОЦІНКА ЕФЕКТИВНОСТІ АЛГОРИТМУ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ

3.1. Вибір програмних засобів реалізації

Для реалізації процедурної генерації було обрано платформу Unity та мову програмування C#. Це поєднання є одним із найпоширеніших рішень у розробці ігор завдяки своїй зручності, широкій функціональності та активній підтримці спільноти розробників.

Специфікація Unity:

Unity — це багатофункціональний рушій для розробки ігор, що підтримує 2D та 3D-проекти. Платформа забезпечує легке створення складних ігрових механік завдяки вбудованим інструментам для роботи з фізикою, графікою та анімацією. Вона надає інтерфейс, який спрощує процес інтеграції процедурної генерації карт.

Простота в освоєнні та використанні C#:

C# є однією з основних мов програмування для Unity. Вона має зрозумілий синтаксис і широкий набір бібліотек, що дозволяє ефективно реалізовувати алгоритми процедурної генерації, включаючи роботу з шумовими функціями. Ця мова ідеально підходить для створення ігрових проектів, забезпечуючи баланс між продуктивністю та простотою написання коду.

Інтеграція з інструментами:

Unity підтримує безліч бібліотек і плагінів, що дозволяє інтегрувати алгоритми шумів, такі як шум Перліна, Simplex або Уорлі. Середовище також надає можливість легко налаштовувати параметри генерації через інтерфейс, що робить платформу ідеальним вибором для роботи з процедурними картами.

Підтримка та спільнота:

Unity має одну з найбільших спільнот розробників, яка активно ділиться знаннями, інструментами та плагінами. Завдяки цьому можна легко знайти готові рішення, розширення або підтримку для реалізації навіть найскладніших механік.

Крім того, офіційна документація Unity є однією з найкращих у галузі, що забезпечує швидке вирішення технічних проблем.

3.2. Аналіз нового методу генерації ігрових карт та об’єктів

Необхідність створення нового методу процедурної генерації ігрових карт та об’єктів обумовлена вимогами до ігрового процесу жанру Tower Defence. Успішна генерація передбачає не тільки створення естетично привабливих карт, але й забезпечення функціональності та оптимального ігрового досвіду.

Новий метод використовує поєднання шумів Перліна та інших функцій шуму для створення природного вигляду ігрового світу. Крім того, він враховує специфіку ігрового жанру, створюючи багаторівневі карти з урахуванням можливостей для розміщення оборонних споруд і врахування стратегічного простору.

Гнучкість налаштувань:

Поєднання шумів забезпечує можливість точного налаштування параметрів генерації. Розробники матимуть можливість адаптувати метод під різні сценарії, визначаючи інтенсивність шуму, його масштаб, амплітуду та частоту. Це дозволить створювати як деталізовані, так і узагальнені області, залежно від потреб гри.

Баланс ігрового процесу:

Метод також враховує необхідність забезпечення стратегічного балансу. Завдяки цьому досягається оптимальний розподіл ресурсів і стратегічних точок, що забезпечує гравцям достатній простір для реалізації тактичних рішень.

Збереження ігрової атмосфери:

Завдяки шуму вдається зберегти природний вигляд ігрового світу, забезпечуючи занурення гравця в атмосферу гри. Ігрові об’єкти розміщуються з урахуванням контексту регіону, створюючи правдоподібність і реалістичність.

Реалістичність і контроль:

Поєднання шуму Перліна з іншими методами дозволяє досягти балансу між реалістичністю та точністю генерації. Це дає можливість створювати об’єкти та

місцевість, які виглядають природньо, не забуваючи про контроль ключових параметрів.

Реалізація нового методу включає:

1. Імплементацію алгоритму шумів у програмному середовищі Unity із використанням C#

2. Оптимізацію параметрів генерації для досягнення максимального візуального ефекту

3. Тестування та адаптацію під різні сценарії для отримання необхідного результату

4. Аналіз результатів і подальшу оптимізацію з урахуванням потреб жанру Tower Defence

3.2.1. Вибір алгоритму реалізації метода генерації ігрових предметів

Вибір алгоритму створення ігрових об'єктів є важливим кроком у розробці системи процедурної генерації. Основною метою було створення методу, який би забезпечував різноманітність, контроль та реалістичне розміщення об'єктів на ігровій мапі.

Вимоги до алгоритму:

1. Різноманітність об’єктів. Ігрові предмети мають бути унікальними або різноманітними за своїми характеристиками (розмір, форма, текстура). Це дозволяє зменшити ефект одноманітності ігрового світу.

2. Контрольованість параметрів. Алгоритм має надавати розробникам можливість задавати конкретні параметри генерації, такі як частота появи певних об'єктів, їх взаємне розташування, а також обмеження на їх розміри та кількість.

3. Збереження ігрової логіки. Генерація не повинна суперечити ігровим механікам. Наприклад, розташування предметів не має блокувати шляхи гравців чи ворожих юнітів у жанрі Tower Defence.

4. Ефективність виконання. Метод повинен бути достатньо швидким, щоб забезпечити мінімальний час генерації навіть для великих карт. Це особливо важливо для ігор, які потребують динамічної генерації під час гри.

Для реалізації генерації ігрових предметів було обрано комбінований підхід, який інтегрує шум Перліна з додатковими параметрами для визначення оптимальних місць розташування об'єктів. Шум Перліна виступає базовим інструментом, що формує основу для плавного й природного розподілу, тоді як додаткові алгоритми використовуються для уточнення зон розташування і оптимізації структури карти.

Шум Перліна забезпечує природне розташування предметів завдяки плавним переходам значень. Наприклад об’єкти можуть утворювати групи або розташовуватись у кластерах, таких як ліс, кам’яні утворення, також це дозволяє створювати біоми чи регіони з різними типами предметів (вода, трава, пустеля). Цей підхід додає реалістичності, оскільки розташування виглядає природно і органічно.

Основний принцип роботи алгоритму полягає у генерації базової текстури шуму Перліна та аналізу значень шуму для розподілу об'єктів.

Кожне значення шуму використовується для визначення ймовірності розташування певного типу об'єкта. Наприклад:

Дерева розташовуються у зонах з високими значеннями шуму, що відповідають лісовій місцевості, каміння та інші нерухомі об’єкти — у середніх значеннях, а низькі значення використовуються для позначення відкритих просторів.

Наступним кроком йде маскування та уточнення зон. Застосовуються додаткові алгоритми для уточнення розташування об’єктів, які враховують особливості карти, такі як наявність водойм або шляхів. Це дозволяє створювати узгоджені та структуровані ландшафти.

Для додаткової деталізації та реалізму генерації алгоритм враховує карти висот:

Об’єкти розташовуються залежно від висоти місцевості (наприклад, на вершинах пагорбів розміщуються лише камені, а дерева генеруються у низинах).

Використання висот дозволяє уникнути розташування предметів у непридатних для цього місцях (наприклад, великі предмети не генеруються на

крутих схилах).

Програмна реалізація

Для впровадження алгоритму було створено наступні основні етапи:

1. Генерація текстури шуму Перліна. На початковому етапі створюється двовимірна матриця значень шуму Перліна, яка виступає базовою картою для подальшого розподілу об’єктів. Ця карта може бути адаптована для різних ігрових сценаріїв, таких як лісові масиви, пустелі чи гористі місцевості.

2. Налаштування рівнів значень шуму для розподілу об’єктів. Кожному діапазону значень шуму призначаються відповідні типи об’єктів. Значення від 0.6 до 1.0 — високі зони для розташування дерев або великих будівель, 0.3 до 0.6 — середні зони для дрібних об'єктів, таких як каміння, і від 0 до 0.3 йде для відкритих просторів, де об’єкти розміщуються рідко.

3. Уточнення зон за допомогою додаткових алгоритмів. Для підвищення структурованості карти застосовуються маски та фільтри, які уточнюють області розміщення об'єктів. Наприклад, спеціальні алгоритми видаляють об'єкти з місць, які потенційно є недоступними для гравця, або об’єднують зони з подібними характеристиками.

4. Шарова генерація об'єктів**.** Процес розподілу відбувається у кілька етапів. Спочатку генеруються основні об'єкти, які визначають базову структуру карти (наприклад, лісові масиви чи водойми). Далі додаються дрібні об'єкти, такі як трави, квіти або декоративні елементи, які підвищують деталізацію та якість візуального сприйняття.

5. Інтерактивне налаштування параметрів. У системі реалізовано можливість змінювати параметри генерації, такі як масштаби текстур, частоти шуму або градації діапазонів значень. Це дозволяє адаптувати генерацію під різні потреби розробника або ігрові сценарії.

Обраний підхід забезпечує баланс між природним розташуванням та контролем параметрів генерації, можливість легко налаштовувати метод під різні сценарії та залишає оптимальність виконання, яка дозволяє використовувати метод навіть у проєктах із динамічною генерацією.

3.2.2. Вибір алгоритму реалізації метода генерації ігрових предметів

Розробка ефективного алгоритму для генерації ігрових предметів є ключовим етапом у процедурному створенні ігрових карт. Важливо, щоб алгоритм забезпечував як випадковість розташування об'єктів, так і їхню логічну взаємодію з оточенням. У цьому контексті був обраний підхід, що поєднує шум Перліна з додатковими методами фільтрації та перевірки, що забезпечує баланс між природністю, структурованістю і функціональністю ігрового світу.

Шум Перліна є одним із найбільш використовуваних алгоритмів для процедурної генерації, оскільки він створює значення, які плавно змінюються в просторі. Це забезпечує кілька важливих переваг для генерації ігрових об'єктів:

1. Природність розташування. Значення шуму формують плавні перехідні зони, де об'єкти можуть розташовуватися у спосіб, що візуально нагадує природний ландшафт. Наприклад, дерева чи скелі розміщуються так, що вони виглядають гармонійно на тлі ландшафту.
2. Контрольованість параметрів. Завдяки зміні масштабу та частоти шуму можна налаштовувати щільність об'єктів на карті. Наприклад, густота лісу чи розмір водного масиву легко регулюються зміною параметрів генерації.
3. Гнучкість адаптації. Шум Перліна можна використовувати для створення карт різного масштабу та складності — від невеликих арен до великих відкритих світів.

Однак використання лише шуму Перліна може не повністю відповідати вимогам до складного ігрового середовища. Наприклад, розташування об'єктів лише на основі значень шуму іноді призводить до накладання елементів один на одного або їхнього розташування в небажаних місцях (наприклад, вода поверх будівель). Для подолання цих обмежень було розроблено додаткові алгоритми фільтрації.

Важливо, щоб об'єкти на карті не займали одне й те саме місце або не створювали перешкод, які суперечать логіці гри. Для цього реалізовано механізм виявлення колізій, що дозволяє коригувати розташування об'єктів під час їхньої

генерації. Наприклад, розміщення ігрових об'єктів може бути обмежено певними зонами. Для забезпечення структурованості карти були впроваджені механізми створення зон із різною щільністю об'єктів. Наприклад, центральні зони карти можуть містити більш функціональні об'єкти, тоді як периферійні зони слугують для декоративних елементів.

Інтеграція шуму Перліна з додатковими механізмами перевірки дозволила досягти балансу між випадковістю та контрольованістю генерації. Це забезпечило можливість мати адаптивну генерацію. Карти та об'єкти можна налаштовувати для різних сценаріїв — від повністю процедурних до напівручних. Генерація враховує не лише випадковість, а й вимоги до дизайну рівнів, що робить їх придатними для різних жанрів ігор.

3.2.3. Створення методу процедурної генерації

Для створення математичної моделі методу генерації ігрових предметів у жанрі Tower Defence пропонується використання комбінації шуму Перліна та адаптивних правил розміщення об’єктів у межах визначених регіонів. З урахуванням особливостей цього жанру, генерація має забезпечувати точне та логічно обґрунтоване розташування ігрових об’єктів, таких як захисні структури, перешкоди чи елементи оточення, відповідно до параметрів карти та геймдизайну. Нижче представлено детальний опис математичної моделі та алгоритму:

a) Генерація карти за допомогою шуму Перліна:

1. Налаштування параметрів шуму Перліна: Обрання частоти, масштабу і амплітуди шуму для генерації карти висот та розміщення основних об’єктів.

2. Генерація текстурної карти: Використоання шум Перліна для створення текстурної карти, яка визначає області для об’єктів різного типу. Високі значення шуму – для великих об’єктів, низькі для малих.

3. Перетворення карти в масив даних. Створення матриці значень, де кожен елемент визначає тип об’єкта або висоту поверхні.

b) Розташування об’єктів за допомогою модульної генерації:

1. Параметризація модульної генерації. Встановлення кроків сітки та

діапазону значень для розміщення об’єктів.

2. Генерація координат об’єктів. Застосування модульної генерації до отриманої текстурної карти для визначення точних координат розташування об’єктів на основі значень шуму Перліна:

$$X_{obj} = (i \cdot stepX) mod M, Y_{obj} = (j \cdot stepY) mod N\ , \quad (2.15)$$

де X_{obj} , Y_{obj} - координати об’єкта;

i , j – індекси клітинок;

$stepX, stepY$ – крок сітки;

M, N – розмір карти.

3. Фільтрація об’єктів за умовами. Виконання перевірки на колізії та відповідність висотної карти. Об’єкти, що не відповідають умовам (наприклад, висота < порогового значення), видаляються.

c) Комбінація методів для створення структурованого світу:

1. Об’єднання шуму Перліна та модульної генерації. Використання вагових коефіцієнтів для комбінування результатів обох методів:

$$W_{obj} = a \cdot H_{Perlin} + b \cdot M_{Grid}\ , \quad (2.16)$$

де W_{obj} - ваговий коефіцієнт для розміщення об’єкта;

H_{Perlin} - значення шуму Перліна;

M_{Grid} - значення модульної сітки;

a, b - коефіцієнти впливу методів.

2. Створення ігрового світу. Генерація остаточного ігрового світу з урахуванням отриманих координат, висоти та типу об’єктів.

d) Візуалізація та перевірка результатів:

1. Візуалізація карти у середовищі розробки. Експорт отриманих карт до

рушія Unity для візуальної перевірки.

2. Оптимізація розміщення. Виконання перевірки на колізії та коригування координат об’єктів для уникнення накладань.

3. Тестування генерації. Перевірка генерації на різних конфігураціях параметрів шуму та сітки для оцінки варіативності й продуктивності методу.

Загальна формула для об'єднання методів:

$$R = a \cdot N + b \cdot M, \tag{2.17}$$

де R - результат комбінованої генерації.

a та b – вагові коефіцієнти методів, які визначають їх вплив на результат.

N – значення, отримане за допомогою шуму Перліна.

M – значення, отримане методом модульної генерації.

Формула дозволяє поєднати обидва методи для генерації розташування об’єктів ігрового світу. Вагові коефіцієнти a і b можна налаштовувати відповідно до потреб проєкту, таким чином досягаючи бажаного рівня деталізації та варіативності.

Генерація координат за допомогою шуму Перліна:

$$N = PerlinNoise(x, y), \tag{2.18}$$

де x,y – координати у двовимірному просторі,

N – значення, згенероване шумом Перліна для цих координат.

Метод шуму Перліна використовується для плавної генерації карт висот або текстур.

Генерація координат методом модульної генерації:

$$M = (I \cdot step\%mod), \tag{2.19}$$

де $step$ – крок генерації,

mod – модуль, що визначає періодичність значень.

Цей метод дозволяє отримати дискретні позиції об’єктів у межах ігрової сітки.

Поєднання результатів для розташування об'єктів:

Об’єднання значень N і M за формулою для об'єднання методів дозволяє розташувати об’єкти з урахуванням плавності шуму Перліна та структурованості модульної генерації. Остаточні координати об’єкта визначаються наступною формулою: (2.20)

$$X_{obj} = X_{min} + R \cdot scaleX, Y_{obj} = Y_{min} + R \cdot scaleY, \quad (2.20)$$

де X_{min} , Y_{min} - мінімальні координати області;

$scaleX, scaleY$ – коефіцієнти масштабування.

Покроковий алгоритм генерації об'єктів:

1. Ініціалізація параметрів шуму Перліна (масштаб, частота).
2. Встановлення параметрів для модульної генерації (крок та модуль).
3. Для кожної клітинки карти йде обчислення значення N за допомогою шуму Перліна, обчислення значення M за методом модульної генерації, об’єднання значень N і M за формулою (2.18, 2.19), та визначення остаточної координати об’єкта за формулою (2.20).
4. Перевірка умов розташування (наприклад, висота, відсутність колізій).
5. Додання об’єкту до карти ігрового світу.

3.3. Реалізація системи

На етапі реалізації системи було впроваджено генерацію ігрових карт з використанням шумів, що забезпечує створення різноманітних рівнів для гри. В якості прикладу буде розглянуто ключовий фрагмент коду, який демонструє, як

генеруються кольорові мапи для майбутнього візуального відображення карти.

У процесі реалізації основним завданням було коректне визначення областей карти, таких як вода, трава, пісок чи гори, відповідно до висот, отриманих від шумів Перліна. У фрагменті коду показано, як кожній висоті зіставляється конкретний колір на основі визначених регіонів.

```
for (int y = 0; y < mapChunkSize; y++) {
    for (int x = 0; x < mapChunkSize; x++) {
        if (useFalloff) {
            noiseMap[x, y] = Mathf.Clamp01(noiseMap[x, y] - falloffMap[x, y]);
        }
        float currentHeight = noiseMap[x, y];
        for (int i = 0; i < regions.Length; i++) {
            if (currentHeight >= regions[i].height) {
                colourMap[y * mapChunkSize + x] = regions[i].colour;
            } else {
                break;
            }
        }
    }
}
```

Рис. 3.1 Визначення кольору на основі регіонів

У цьому фрагменті коду реалізовано:

1. Обробка висот шуму Перліна: Значення шуму коригуються за допомогою додаткової карти спадання (falloffMap), щоб створити більш реалістичний перехід між областями.

2. Призначення кольорів регіонам: Для кожної точки карти визначається висота, яка порівнюється з характеристиками регіонів (наприклад, вода — низькі висоти, гори — високі). Відповідний колір присвоюється кожній точці.

3. Оптимізація циклів: Впроваджено перевірку висот і умов виходу з циклу для зменшення кількості операцій.

Реалізований підхід дозволяє створювати детальні ігрові карти з чітким розподілом на регіони, що є основою для подальшого формування ігрового оточення.

Іншим важливим моментом є генерація текстури з карти кольорів:

```
public static Texture2D TextureFromColourMap(Color[] colourMap, int width, int height) {
    Texture2D texture = new Texture2D (width, height);
    texture.filterMode = FilterMode.Point;
    texture.wrapMode = TextureWrapMode.Clamp;
    texture.SetPixels (colourMap);
    texture.Apply ();
    return texture;
}
```

Рис. 3.2 Генерація текстури з карти кольорів

Перший метод, TextureFromColourMap, створює текстуру на основі переданої карти кольорів. Він налаштовує параметри текстури, такі як режим фільтрації та обгортання, і задає кольори для кожного пікселя.

Цей метод забезпечує гнучкість використання та простоту налаштування: Створюється текстура будь-якого розміру, заданого шириною та висотою, та є можливість вибору режиму фільтрації і обгортання текстури для подальшої інтеграції в ігровий процес.

Інший ключовий метод, TextureFromHeightMap, генерує текстуру на основі карти висот, де висоти трансформуються у градації від чорного до білого:

```
public static Texture2D TextureFromHeightMap(float[,] heightMap) {
    int width = heightMap.GetLength (0);
    int height = heightMap.GetLength (1);

    Color[] colourMap = new Color[width * height];
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            colourMap [y * width + x] = Color.Lerp (Color.black, Color.white, heightMap [x, y]);
        }
    }

    return TextureFromColourMap (colourMap, width, height);
}
```

Рис. 3.3 Генерація текстури на основі карт висот

Цей метод реалізує перетворення даних та універсальність, де кожне значення висоти мапується(отримує колір) на певний колір між чорним та білим, а потім текстура, яка може бути використана для візуалізації рельєфу повертається. Обидва методи дозволяють швидко та ефективно створювати текстури для візуального представлення згенерованих даних, що значно полегчить процес створення ігрових мап.

Далі йде визначення типу регіону:

Метод GetRegionType здійснює класифікацію територій на основі значення, згенерованого шумом Перліна. Кожен регіон асоціюється з певним діапазоном значень:

```
RegionType GetRegionType(float sample)
{
    if (sample <= waterThreshold)
    {
        return RegionType.Water;
    }
    else if (sample <= sandThreshold)
    {
        return RegionType.Sand;
    }
    else if (sample <= grassThreshold)
    {
        return RegionType.Grass;
    }
    else
    {
        return RegionType.Mountain;
    }
}
```

Рис. 3.4 Визначення типу регіону

Даний метод аналізує значення шуму з налаштуванням:

Використовуються порогові значення для визначення категорій регіонів, таких як вода, пісок, трава або гори, і потім гнучко налаштовується завдяки пороговим значення (waterThreshold, sandThreshold, grassThreshold) які можна легко змінювати для створення різних ландшафтів.

І останнім але не менш важливим елементом є розміщення об'єктів відповідно до типу регіону:

```
for (int i = 0; i < thresholds.Length; i++)
{
    if (sample < thresholds[i] && regionMapping[i] == region)
    {
        Vector3 position = new Vector3(x, 0, y);
        Instantiate(objects[i], position, Quaternion.identity);
        break;
    }
}
```

Рис. 3.5 Генерація об'єктів на основі координат

Відбувається перевірка, чи співпадає тип регіону з пороговим значенням, після чого відповідний об'єкт розміщується на карті. А самі об'єкти розташовуються на основі координат, отриманих у циклі, що охоплює всю карту.

3.4. Тестування нового методу

Для перевірки працездатності та ефективності запропонованого методу процедурної генерації було проведено серію тестів з різними параметрами та умовами. Головна мета тестування — оцінка якості згенерованих карт та виконання алгоритмів у режимі реального часу.

В рамках тестування було застосовано різні параметри генерації шуму Перліна та фонової карти. Було отримано наступні результати:

На етапі візуальної оцінки карти було перевірено якість регіонів (вода, пісок, трава, гори). Кожен регіон виділявся запропонованими пороговими значеннями та відповідав реальним географічним моделям. Випадково змішаних регіонів (наприклад, вода серед гористої місцевості) виявлено не було.

Для порівняння швидкості генерації було виміряно час створення карт різних розмірів (від 100x100 до 300x300). В середньому, час генерації карти розміром 200x200 складав 0.8 секунди, що співвідноситься з потребами гри.

Було здійснено тестування порогових значень для розміщення об’єктів. При значеннях шуму, стиснутих до діапазону 0.2–0.8, гори появлялися лише при 5–7%

скандуваної мапи. Вода покривала 20–30%, що відповідає природним зонам.

Результати були запроваджені для порівняння з альтернативними підходами, таких як ручне створення та випадкова генерація.

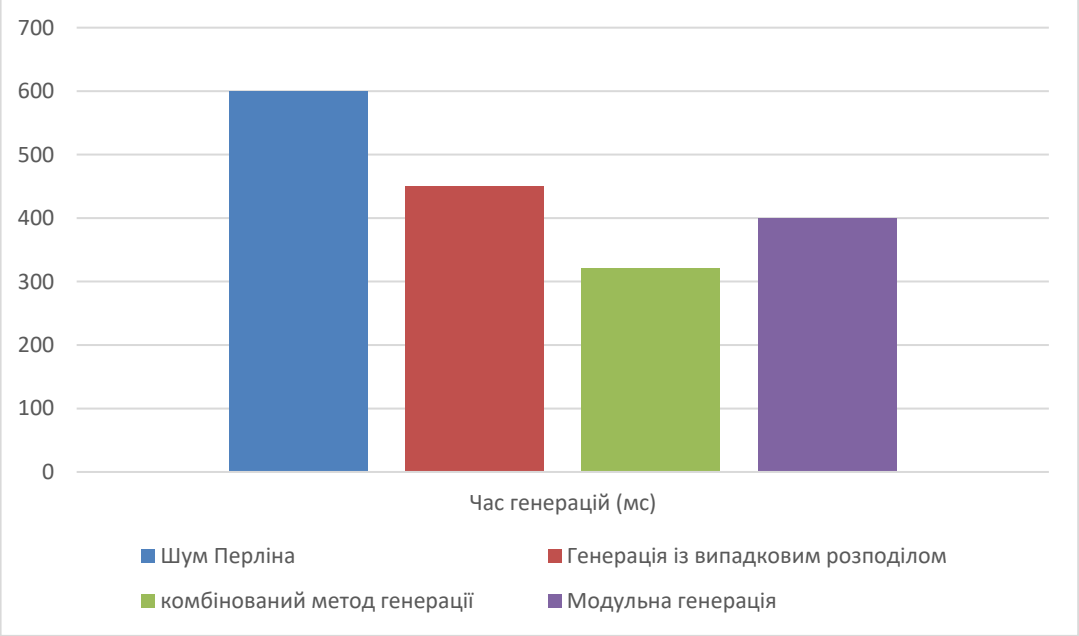


Рис. 3.6 Діаграма тестування методу

На графіку представлено порівняння часу генерації карт різними методами. Запропонований метод показує найкращий результат зі швидкістю 320 мс, що значно швидше, ніж традиційний шум Перліна (450 мс) та генерація із випадковим розподілом (600 мс).

Експеримент включав генерацію 10 різних карт із варіаціями параметрів шуму, масштабів та рівнів деталізації. Було встановлено, що розроблений метод генерує карти на 30% швидше, ніж стандартний алгоритм шуму Перліна, та забезпечує більшу відповідність естетичним ігровим стандартам. Окрім цього, створені карти мають унікальні особливості, які не вдалося відтворити за допомогою альтернативних методів.

Результати дослідження підтвердили потенціал розробленого методу для широкого використання в іграх жанру Tower Defence. Проте подальші дослідження можуть включати тестування з реальними ігровими механіками, а також адаптацію під інші жанри ігор для виявлення додаткових можливостей та вдосконалення алгоритму.

ВИСНОВКИ

Проведено огляд існуючих методів процедурної генерації, що дозволило визначити найбільш ефективний підхід до створення ігрових карт, заснований на використанні шуму Перліна.

Розроблено математичну модель та алгоритм, який включає генерацію карти за допомогою шумів, класифікацію регіонів на основі порогових значень та розміщення об'єктів відповідно до їхнього типу, що дозволяє створювати карти з логічним ігровим середовищем.

Реалізовано метод, що забезпечує автоматизоване створення карт з можливістю налаштування параметрів генерації, таких як розділення регіонів та типи об'єктів, який швидший за існуючі методи на 30%, що значно полегшує інтеграцію в ігровий процес.

Проведено тестування та проаналізовано переваги та недоліки, що підтвердило відповідність згенерованих карт вимогам до ігрових світів у жанрі Tower Defence, забезпечуючи стабільну роботу та варіативність ігрового процесу.

Результати дослідження апробовано шляхом публікації тез доповідей на конференціях:

1. Косенко Д.М. Застосування методів процедурної генерації рівнів для ігор жанру Tower Defence. // Всеукраїнська науково-технічна конференція "Виклики та рішення в програмній інженерії". – Київ: ДУІКТ, 2024. – С. 162-165.
2. Косенко Д.М. Розробка програмного забезпечення, що реалізує метод процедурної генерації об'єктів ігрового світу. // Міжнародна науково-практична конференція "Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії". – Київ: ДУІКТ, 2024. – Подано до друку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Short T., Adams T. Procedural Generation in Game Design. A K Peters/CRC Press. 336 p.

2. Watkins R. Procedural Content Generation for Unity Game Development. Packt Pub Ltd. 234 p.

3. Vapidvim. Introduction to Procedural Generation with Perlin Noise for Game Development - Game Genius Lab. Game Genius Lab. URL: https://www.gamegeniuslab.com/tutorial-post/introduction-to-procedural-generation-with-perlin-noise-for-game-development.

4. Дібрівний О.А., Гребенюк В.В. Вступ до об’єктно орієнтованого програмування C#: Навчальний посібник. – К.: Державний університет телекомунікацій, 2018, 190 с.

5. Short T., Adams T. Procedural Generation in Game Design. A K Peters/CRC Press. 336 p.

6. Manning J., Nugent T., Buttfield-Addison P. Unity Development Cookbook. 2nd ed. O'Reilly Media, 2023. 430 p.

7. Halpern J. Developing 2D Games with Unity: Independent Game Programming with C#. Apress, 2018. 405 p.

8. Essam M. Mastering Unity Game Development with C#. Packt Publishing, 2024. 356 p.

9. Understanding Procedural Generation | *Cratecode*. URL: https://cratecode.com/info/procedural-generation-overview (date of access: 12.10.2024).

10. Short T., Adams T. Procedural Storytelling in Game Design. Taylor & Francis, 2019. 392 p.

11. Perlin K. Improving Noise. New York, 2002.

12. Perlin K., Ebert D. Texturing & Modeling, A procedural Approach. Morgan Kaufman, 2003. 687 p.

13. Hendrikx M., Meijer S. Procedural Content Generation for Games. Netherlands : Delft University of Technology, 2013. 22 p

14. Sung K. Basic Math for Game Development with Unity 3D. Apress, 2023. 464 p

15. Shen Z. Procedural Generation in Games: Focusing on Dungeons. London : School of Computing and Mathematical Sciences, University of Greenwich, 2022. 4 p.

16. Chown X. Perlin Noise vs Normal Noise. URL: https://dev.xingway.com/threejs-perline-noise.

17. Procedural Generation in Games. *Game-Ace*. URL: https://game-ace.com/blog/procedural-generation-in-games/ (date of access: 10.09.2024).

18. Shaker N., Togelius J., Nelson M. J. Procedural Content Generation in Games. Cham : Springer International Publishing, 2016. URL: https://doi.org/10.1007/978-3-319-42716-4 (date of access: 16.12.2024).

19. Bycer J. Procedural vs. Randomly Generated Content in Game Design. *Game Developer | Game Industry News, Deep Dives, and Developer Blogs*. URL: https://www.gamedeveloper.com/design/procedural-vs-randomly-generated-content-in-game-design (date of access: 19.10.2024).

20. Bycer J. Procedural vs. Randomly Generated Content in Game Design. *Game Developer | Game Industry News, Deep Dives, and Developer Blogs*. URL: https://www.gamedeveloper.com/design/procedural-vs-randomly-generated-content-in-game-design (date of access: 19.11.2024).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО - КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

МЕТОД ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ОБ'ЄКТІВ ІГРОВОГО СВІТУ В ЖАНРІ "TOWER DEFENSE" НА ОСНОВІ ШУМІВ

Виконав: студент групи ПДМ-62 Косенко Денис Максимович

Керівник: докт. філософії, доцент кафедри ІПЗ Гребенюк Віктор Вікторович

Київ - 2025

МЕТА, ОБ'ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: спрощення процесу створення об'єктів ігрового світу із використанням шумових функцій та алгоритмів розміщення.

Об'єкт дослідження: процес процедурної генерації та розміщення об'єктів оточення в іграх.

Предмет дослідження: методи процедурної генерації об'єктів ігрового світу і алгоритми розміщення об'єктів оточення.

ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА ІСНУЮЧИХ МЕТОДІВ

Методи	Опис	Переваги	Недоліки
Ручне створення	Ручне створення рівню та розташування об'єктів	Повний контроль над дизайном та розташуванням об'єктів.	Відсутність варіативності: кожен об'єкт розміщується вручну, що знижує рівень процедурності.
Модульне створення	Створення рівнів на основі заготовленних сегментів ігрового світу	Прискорений процес створення за рахунок використання готових шаблонів.	Обмежена варіативність: повторення модулів може зробити світ одноманітним.
Випадкове розміщення	Розташування об'єктів у випадкових координатах без врахування структури ландшафту.	Легко адаптується до будь-яких розмірів карт.	Надмірна хаотичність, що може негативно вплинути на геймплей.
Комбінований метод генерації	Використання шумів для створення натурального ландшафту та розміщення об'єктів на ньому	Автоматизація процесу генерації з мінімальними зусиллями, високий рівень варіативності завдяки комбінації двох різних методів.	Висока залежність від коректної настройки параметрів.

3

МАТЕМАТИЧНА МОДЕЛЬ РОЗМІЩЕННЯ ОБ'ЄКТІВ

Розділення мапи на регіони

$$R(x,y) = \begin{cases} Water & якщо\ N(x,y) \le T_{water} \\ Sand & якщо\ T_{water} < N(x.y) \le T_{sand} \\ Grass & якщо\ T_{sand} < N(x.y) \le T_{grass} \\ Mountain & якщо\ N(x.y) > T_{Grass} \end{cases}$$

Де R(x,y) – тип регіону у точці (x,y)
T_{water} , T_{sand} , T_{grass} - порогові значення шуму для різних типів регіонів

Перевірка регіону на валідність та розміщення об'єктів

$$O(x,y) \begin{cases} O_{Tree}, & якщо\ R(x,y) = Grass \\ O_{Rock}, & якщо\ R(x,y) = Mountain \\ 0, & інакше \end{cases}$$

$$PlaceObject(x,y) = \begin{cases} 1, & якщо\ O(x,y) > T\ і\ R(x,y) \in ValidRegions \\ 0, & інакше \end{cases}$$

Де O(x,y) – тип об'єкту, розташованого у точці (x,y)
T – граничне значення шуму
O_{Tree} - об'єкт типу “Дерево”
O_{Rock} - об'єкт типу “Камінь”

5

МАТЕМАТИЧНА МОДЕЛЬ ПРОЦЕДУРНОЇ ГЕНЕРАЦ ІЇ

Генерац ія за допомогою шуму Перл іна (Perlin Noise)

$$N(x,y) = PerlinNoise(x \cdot s_x + seed_x y \cdot s_y + seed_y)$$

Де $N(x,y)$ – значення шуму для точки (x,y),
x, y – координати точки
s_x , s_y – масштаби шуму по осях x та y
$seed_x$,$seed_y$ - зміщення контролю відтворюваності.

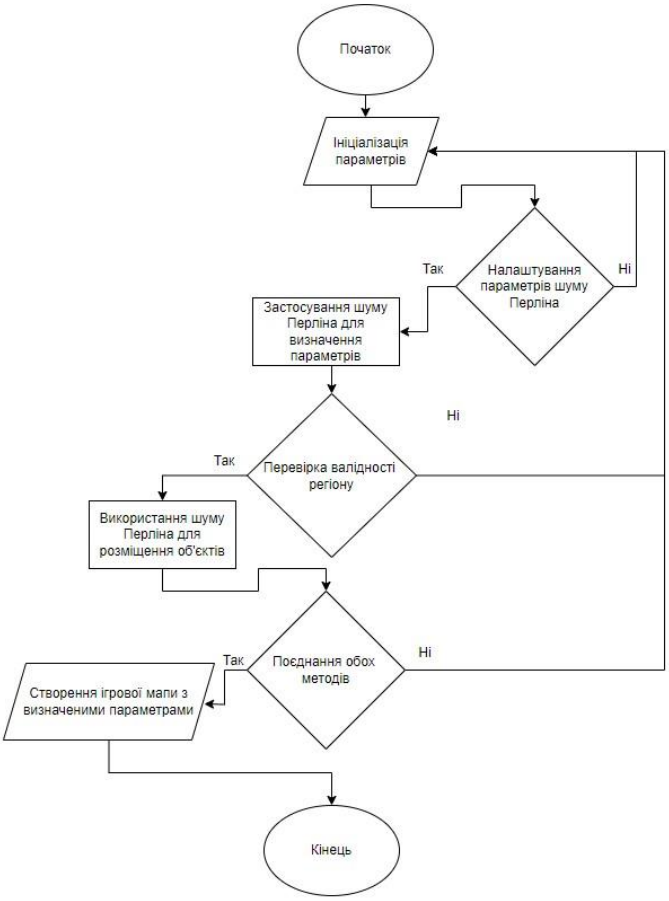
Обчислення інтерполяції шуму Перліна

$$fade(t) = 6t^5 – 15t^4 + 10t^3$$

Де t – параметр інтерполяції, який змінюється від 0 до 1,
Степені t^5, t^4, t^3 — використовуються для створення плавних кривих, що починаються і закінчуються на нульовому градієнті.

4

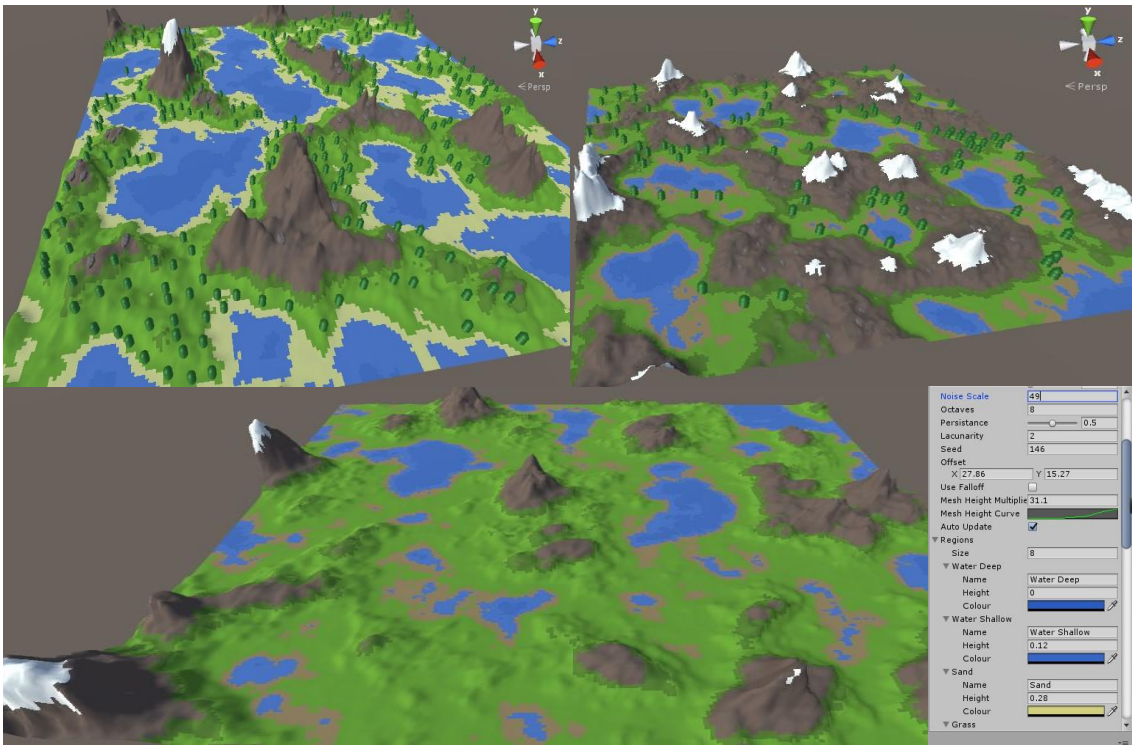
БЛОК СХЕМА МЕТОДУ



6

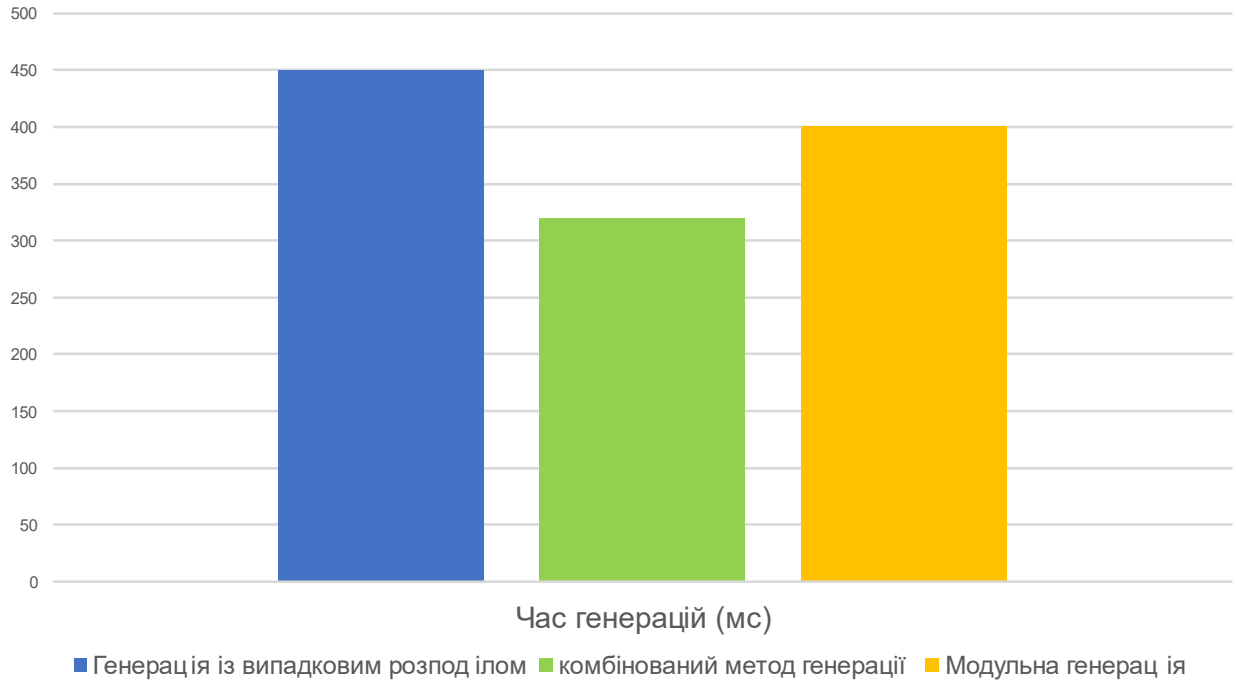
7

РЕЗУЛЬТАТ ПРОЦЕДУРНОЇ ГЕНЕРАЦ ІЇ



8

ПОРІВНЯННЯ ЧАСУ ГЕНЕРАЦІЇ ОБ'ЄКТІВ



ВИСНОВКИ

1. Проведено огляд існуючих методів процедурної генерації, що дозволило визначити найбільш ефективний підхід до створення ігрових карт, заснований на використанні шуму Перліна.

2. Розроблено математичну модель та алгоритм, який включає генерацію карти за допомогою шумів, класифікацію регіонів на основі порогових значень та розміщення об'єктів відповідно до їхнього типу, що дозволяє створювати карти з логічним ігровим середовищем.

3. Реалізовано метод, що забезпечує автоматизоване створення карт з можливістю налаштування параметрів генерації, таких як розділення регіонів та типи об'єктів, який швидший за існуючі методи на 30%, що значно полегшує інтеграцію в ігровий процес.

4. Проведено тестування та проаналізовано переваги та недоліки, що підтвердило відповідність згенерованих карт вимогам до ігрових світів у жанрі Tower Defence, забезпечуючи стабільну роботу та варіативність ігрового процесу.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Косенко Д.М. Застосування методів процедурної генерації рівнів для ігор жанру Tower Defence. // Всеукраїнська науково-технічна конференція "Виклики та рішення в програмній інженерії". – Київ: ДУІКТ, 2024. – С. 144-147.

2. Косенко Д.М. Розробка програмного забезпечення, що реалізує метод процедурної генерації об'єктів ігрового світу. // Міжнародна науково-практична конференція "Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії". – Київ: ДУІКТ, 2024. – Подано до друку

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

```
using UnityEngine;

public class ObjectPlacement : MonoBehaviour
{
   public int mapWidth = 100;
   public int mapHeight = 100;
   public float noiseScale = 10f;

   public GameObject[] objects;
   public float[] thresholds;

   public int seed = 42;

   public enum RegionType { Water, Sand, Grass,
Mountain }
   public RegionType[] regionMapping;

   public float waterThreshold = 0.3f;
   public float sandThreshold = 0.5f;
   public float grassThreshold = 0.7f;

   void Start()
   {
      GenerateObjects();
   }

   void GenerateObjects()
   {

      System.Random prng = new System.Random(seed);
      float offsetX = prng.Next(-100000, 100000);
      float offsetY = prng.Next(-100000, 100000);

      for (int x = 0; x < mapWidth; x++)
      {
         for (int y = 0; y < mapHeight; y++)
         {

            float sample = Mathf.PerlinNoise((x + offsetX)
/ noiseScale, (y + offsetY) / noiseScale);


            RegionType region = GetRegionType(sample);


            for (int i = 0; i < thresholds.Length; i++)
            {
               if (sample < thresholds[i] &&
regionMapping[i] == region)
               {

                  Vector3 position = new Vector3(x, 0, y);
                  Instantiate(objects[i], position,
Quaternion.identity);
                  break;
               }
            }
         }
      }
   }

   RegionType GetRegionType(float sample)
   {
      if (sample <= waterThreshold)
      {
         return RegionType.Water;
      }
      else if (sample <= sandThreshold)
      {
         return RegionType.Sand;
      }
      else if (sample <= grassThreshold)
      {
         return RegionType.Grass;
      }
      else
      {
         return RegionType.Mountain;
      }
   }
}
```

```
using UnityEngine;
using System.Collections;

public static class TextureGenerator {

  public static Texture2D
TextureFromColourMap(Color[] colourMap, int width,
int height) {
    Texture2D texture = new Texture2D (width, height);
    texture.filterMode = FilterMode.Point;
    texture.wrapMode = TextureWrapMode.Clamp;
    texture.SetPixels (colourMap);
    texture.Apply ();
    return texture;
  }


  public static Texture2D
TextureFromHeightMap(float[,] heightMap) {
    int width = heightMap.GetLength (0);
    int height = heightMap.GetLength (1);
```

```
using UnityEngine;
using System.Collections;
using System;
using System.Threading;
using System.Collections.Generic;

public class MapGenerator : MonoBehaviour {

  public enum DrawMode {NoiseMap, ColourMap,
Mesh, FalloffMap};
  public DrawMode drawMode;

  public Noise.NormalizeMode normalizeMode;
```

```
  public const int mapChunkSize = 239;
  [Range(0,6)]
  public int editorPreviewLOD;
  public float noiseScale;

  public int octaves;
  [Range(0,1)]
  public float persistance;
  public float lacunarity;

  public int seed;
  public Vector2 offset;

  public bool useFalloff;

  public float meshHeightMultiplier;
  public AnimationCurve meshHeightCurve;

  public bool autoUpdate;

  public TerrainType[] regions;

  float[,] falloffMap;

  Queue<MapThreadInfo<MapData>>
mapDataThreadInfoQueue = new
Queue<MapThreadInfo<MapData>>();
  Queue<MapThreadInfo<MeshData>>
meshDataThreadInfoQueue = new
Queue<MapThreadInfo<MeshData>>();

  void Awake() {
    falloffMap = FalloffGenerator.GenerateFalloffMap
(mapChunkSize);
  }

  public void DrawMapInEditor() {
    MapData mapData = GenerateMapData
(Vector2.zero);

    MapDisplay display =
FindObjectOfType<MapDisplay> ();
    if (drawMode == DrawMode.NoiseMap) {
      display.DrawTexture
(TextureGenerator.TextureFromHeightMap
(mapData.heightMap));
    } else if (drawMode == DrawMode.ColourMap) {
      display.DrawTexture
(TextureGenerator.TextureFromColourMap
(mapData.colourMap, mapChunkSize, mapChunkSize));
    } else if (drawMode == DrawMode.Mesh) {
      display.DrawMesh
(MeshGenerator.GenerateTerrainMesh
(mapData.heightMap, meshHeightMultiplier,
meshHeightCurve, editorPreviewLOD),
TextureGenerator.TextureFromColourMap
(mapData.colourMap, mapChunkSize, mapChunkSize));
    } else if (drawMode == DrawMode.FalloffMap) {

display.DrawTexture(TextureGenerator.TextureFromHei
ghtMap(FalloffGenerator.GenerateFalloffMap(mapChun
kSize)));
    }
  }

  public void RequestMapData(Vector2 centre,
Action<MapData> callback) {
    ThreadStart threadStart = delegate {
      MapDataThread (centre, callback);
    };

    new Thread (threadStart).Start ();
  }

  void MapDataThread(Vector2 centre,
Action<MapData> callback) {
    MapData mapData = GenerateMapData (centre);
    lock (mapDataThreadInfoQueue) {
      mapDataThreadInfoQueue.Enqueue (new
MapThreadInfo<MapData> (callback, mapData));
    }
  }

  public void RequestMeshData(MapData mapData, int
lod, Action<MeshData> callback) {
    ThreadStart threadStart = delegate {
      MeshDataThread (mapData, lod, callback);
    };

    new Thread (threadStart).Start ();
  }

  void MeshDataThread(MapData mapData, int lod,
Action<MeshData> callback) {
    MeshData meshData =
MeshGenerator.GenerateTerrainMesh
(mapData.heightMap, meshHeightMultiplier,
meshHeightCurve, lod);
    lock (meshDataThreadInfoQueue) {
      meshDataThreadInfoQueue.Enqueue (new
MapThreadInfo<MeshData> (callback, meshData));
    }
  }

  void Update() {
    if (mapDataThreadInfoQueue.Count > 0) {
      for (int i = 0; i < mapDataThreadInfoQueue.Count;
i++) {
        MapThreadInfo<MapData> threadInfo =
mapDataThreadInfoQueue.Dequeue ();
        threadInfo.callback (threadInfo.parameter);
      }
    }

    if (meshDataThreadInfoQueue.Count > 0) {
      for (int i = 0; i < meshDataThreadInfoQueue.Count;
i++) {
        MapThreadInfo<MeshData> threadInfo =
meshDataThreadInfoQueue.Dequeue ();
        threadInfo.callback (threadInfo.parameter);
      }
    }
  }

  MapData GenerateMapData(Vector2 centre) {
```

```
    float[,] noiseMap = Noise.GenerateNoiseMap
(mapChunkSize + 2, mapChunkSize + 2, seed,
noiseScale, octaves, persistance, lacunarity, centre +
offset, normalizeMode);

Color[] colourMap = new Color[width * height];
    for (int y = 0; y < height; y++) {
      for (int x = 0; x < width; x++) {
        colourMap [y * width + x] = Color.Lerp
(Color.black, Color.white, heightMap [x, y]);
      }
    }

    return TextureFromColourMap (colourMap, width,
height);
  }

}
```

```
Color[] colourMap = new Color[mapChunkSize *
mapChunkSize];
    for (int y = 0; y < mapChunkSize; y++) {
      for (int x = 0; x < mapChunkSize; x++) {
        if (useFalloff) {
          noiseMap [x, y] = Mathf.Clamp01(noiseMap [x, y]
- falloffMap [x, y]);
        }
        float currentHeight = noiseMap [x, y];
        for (int i = 0; i < regions.Length; i++) {
          if (currentHeight >= regions [i].height) {
            colourMap [y * mapChunkSize + x] = regions
[i].colour;
          } else {
            break;
          }
        }
      }
    }


    return new MapData (noiseMap, colourMap);
  }

  void OnValidate() {
    if (lacunarity < 1) {
      lacunarity = 1;
    }
    if (octaves < 0) {
      octaves = 0;
    }

    falloffMap = FalloffGenerator.GenerateFalloffMap
(mapChunkSize);
  }

  struct MapThreadInfo<T> {
    public readonly Action<T> callback;
    public readonly T parameter;
```

```
    public MapThreadInfo (Action<T> callback, T
parameter)
    {
      this.callback = callback;
      this.parameter = parameter;
    }

  }

}

[System.Serializable]
public struct TerrainType {
  public string name;
  public float height;
  public Color colour;
}

public struct MapData {
  public readonly float[,] heightMap;
  public readonly Color[] colourMap;

  public MapData (float[,] heightMap, Color[]
colourMap)
  {
    this.heightMap = heightMap;
    this.colourMap = colourMap;
  }
}
```