ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
(повне найменування вищого навчального закладу)
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
(повне найменування інституту, назва факультету (відділення))
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ І ТЕХНОЛОГІЙ
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до кваліфікаційної роботи магістра
(освітній рівень)

на тему: «Дослідження алгоритмів процедурної генерації ігрових карт та ландшафтів»

Виконав: студент групи 6ПР1
спеціальності
121 - «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

Ільїн Герман Андрійович
(прізвище та ініціали)

Керівник д.т.н., проф. Жарікова М.В.
(прізвище та ініціали)

Рецензент к.т.н., доцент Григорова А.А.
(прізвище та ініціали)

Херсон – 2025

Херсонський національний технічний університет
(повне найменування закладу вищої освіти)

Факультет, відділення Інформаційних технологій та дизайну
Кафедра Програмних засобів і технологій
Освітній рівень Магістр
Спеціальність 121 – Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри
Програмних засобів і технологій
к.т.н. доц. Огнєва О. Є.
“___” _____ 2025 р.

З А В Д А Н Н Я
НА ВИПУСКНУ РОБОТУ СТУДЕНТУ

Ільїну Герману Андрійовичу
(прізвище, ім’я, по батькові)

Тема роботи «Дослідження алгоритмів процедурної генерації ігрових карт та ландшафтів»

керівник роботи д.т.н., проф. Жарікова М.В.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від .. . .. . 2025 р. № -

1. Строк подання студентом роботи _____
2. Вихідні дані до роботи літературні та періодичні джерела, матеріали преддипломної практики, _____
3. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
   a Дослідження та аналіз предметної області
   b Аналіз вимог та розробка проектних специфікацій
   c Проектування програмного продукту
   d Розробка та тестування алгоритмів процедурної генерації
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 23.09.2025

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів виконання роботи	Термін виконання етапів роботи	Примітки
1.	Отримання завдання	23.09.2025	Виконано
2.	Підбір літератури	28.09.2025	Виконано
3.	Аналіз предметної області	05.10.2025	Виконано
4.	Розробка та обґрунтування завдання	13.10.2025	Виконано
5.	Розробка концептуальної моделі	16.10.2025	Виконано
6.	Розробка алгоритму	28.10.2025	Виконано
7.	Проектування програми	06.11.2025	Виконано
8.	Розробка інтерфейсу програми	10.11.2025	Виконано
9.	Тестування програми	17.11.2025	Виконано
10.	Оформлення пояснювальної записки	30.11.2025	Виконано
11.	Захист кваліфікаційної роботи	22.12.2025	Виконано

Студент _____ Ільїн Г.А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Жарікова М.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок – 53 Рисунків – 8 Джерел – 16

Ільїн Г.А. Дослідження алгоритмів процедурної генерації ігрових карт та ландшафтів: кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення». — Херсон.

Дана кваліфікаційна робота присвячена дослідженню та практичній реалізації алгоритмів процедурної генерації ігрових карт і ландшафтів у контексті розробки ігрових середовищ. У сучасній ігровій індустрії, що динамічно розвивається, зростає потреба у створенні варіативних, масштабних і реіграбельних ігрових світів, що неможливо ефективно забезпечити виключно ручними методами проєктування рівнів.

Метою роботи є дослідження алгоритмів процедурної генерації ігрових карт та ландшафтів, а також розробка ігрового середовища з керованою складністю і високою варіативністю. Для досягнення поставленої мети в роботі здійснено аналіз сучасних ігрових проєктів і підходів до генерації контенту, досліджено основні алгоритмічні методи процедурної генерації.

У межах роботи розроблено систему процедурної генерації ігрового середовища з використанням ігрового рушія Unity та мови програмування C#. Згенеровані карти забезпечують логічну структуру простору, варіативність та підтримку навігації автономних агентів.

Практична значущість роботи полягає у можливості використання отриманих результатів під час розробки ігрових застосунків, інтерактивних симуляцій та дослідницьких проєктів, пов’язаних зі створенням віртуальних середовищ. Результати дослідження можуть бути застосовані у сфері ігрового дизайну та подальшого розвитку технологій процедурної генерації контенту.

Ключові слова: ігрове середовище, процедурна генерація, ігрові карти, ландшафти, алгоритми, варіативність, Unity.

ABSTRACT

Pages – 53 Pictures – 8 Sources – 16

Ilin H.A. Research of procedural generation algorithms for game maps and landscapes: Master’s qualification work in specialty 121 "Software Engineering". — Kherson.

This qualification work focuses on the research and practical implementation of procedural generation algorithms for game maps and landscapes within the context of game environment development. In today's dynamically evolving gaming industry, there is a growing demand for creating diverse, large-scale, and replayable game worlds, which cannot be efficiently achieved through manual level design methods alone.

The objective of the work is to study procedural generation algorithms for game maps and landscapes and to develop a game environment with controlled complexity and high variability. To achieve this goal, the study analyzes modern game projects and content generation approaches, and explores the fundamental algorithmic methods of procedural generation.

Within the scope of the work, a procedural generation system for a game environment was developed using the Unity game engine and the C# programming language. The generated maps provide a logical spatial structure, variability, and navigation support for autonomous agents.

The practical significance of the work lies in the possibility of using the obtained results in the development of game applications, interactive simulations, and research projects related to the creation of virtual environments. The research findings can be applied in the field of game design and the further development of procedural content generation (PCG) technologies.

Keywords: game environment, procedural generation, game maps, landscapes, algorithms, variability, Unity.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI (Artificial Intelligence) — штучний інтелект.

БПЛА — безпілотний літальний апарат.

NPC (Non-Player Character) — неігровий персонаж.

PCG (Procedural Content Generation) — процедурна генерація контенту.

Roguelike — жанр комп’ютерних ігор із процедурною генерацією рівнів та високою складністю.

Unity — ігровий рушій для розробки дво- та тривимірних ігор і симуляцій.

C# — об’єктно-орієнтована мова програмування платформи .NET.

Ігрова карта — віртуальне просторове середовище, в якому відбувається ігровий процес.

Ландшафт — сукупність просторових елементів ігрового середовища, що імітують природний або штучний рельєф.

Процедурна генерація — метод автоматичного створення контенту на основі алгоритмів і заданих правил.

Автономний агент — програмний об’єкт, здатний самостійно ухвалювати рішення та взаємодіяти з середовищем.

Геймплей — сукупність ігрових механік і взаємодій гравця з ігровим середовищем.

Реіграбельність — здатність гри залишатися цікавою при повторному проходженні.

Навігація — процес визначення та проходження маршруту в ігровому середовищі.

Domain Randomization — метод варіювання параметрів середовища для підвищення узагальнювальної здатності агентів.

Emergent Gameplay — ігрові ситуації, що виникають спонтанно внаслідок взаємодії гравця з системами гри.

ВСТУП

Сучасна індустрія комп’ютерних ігор та інтерактивних симуляцій активно розвивається у напрямку створення великих, різноманітних і динамічних віртуальних світів. Зі зростанням складності ігрових проєктів та вимог до їх реалістичності традиційні підходи до ручного створення ігрових рівнів і ландшафтів стають малоефективними. Проєктування кожної карти вручну вимагає значних часових та людських ресурсів, а також обмежує варіативність ігрового середовища. У зв’язку з цим особливої актуальності набувають методи процедурної генерації, які дозволяють автоматично створювати ігрові карти та ландшафти на основі формальних алгоритмів і математичних моделей.

Процедурна генерація ігрових середовищ дає змогу не лише суттєво зменшити витрати на розробку контенту, але й створювати унікальний ігровий досвід за рахунок постійної змінюваності карт, рівнів і сцен. Кожен запуск гри або симуляції може супроводжуватися формуванням нового середовища, що підвищує реіграбельність та зацікавленість користувача. Крім того, процедурна генерація відкриває можливості для адаптації складності гри під стиль гри користувача або під поведінку автономних ігрових агентів.

Окрім безпосереднього використання в ігровій індустрії, процедурно згенеровані середовища дедалі частіше застосовуються в інтерактивних симуляторах і системах зі штучним інтелектом. У таких системах ігрові карти слугують тренувальними просторами для автономних агентів, які навчаються навігації, ухваленню рішень і взаємодії з середовищем. До таких агентів можуть належати не лише ігрові персонажі або NPC, але й складні автономні системи, зокрема безпілотні літальні апарати, які використовуються як модель рухомого агента у тривимірному просторі. У даному контексті БПЛА розглядається не як самостійний об’єкт дослідження, а як один із можливих прикладів застосування процедурно згенерованих ігрових карт у симуляційних середовищах.

Актуальність даної роботи зумовлена зростаючою потребою у методах створення ігрових карт і ландшафтів, які поєднують високу варіативність, керовану складність і правдоподібність. Процедурна генерація дозволяє формувати середовища, що імітують як природні ландшафти, так і штучні структури, такі як підземелля, міські простори або складні багаторівневі рівні. Саме такі середовища є найбільш цікавими з погляду ігрового процесу та поведінки автономних агентів.

Метою даної дипломної роботи є дослідження алгоритмів процедурної генерації ігрових карт та ландшафтів, а також аналіз можливостей їх використання в інтерактивних ігрових середовищах і симуляторах автономних агентів. Для досягнення поставленої мети в роботі необхідно розв’язати такі завдання:

- проаналізувати теоретичні основи процедурної генерації карт і ландшафтів;
- дослідити математичні та алгоритмічні підходи до створення ігрових середовищ;
- виконати аналіз існуючих методів генерації з точки зору їх впливу на структуру та складність ігрових карт;
- розробити власну систему процедурної генерації ігрових середовищ;
- оцінити можливість використання згенерованих карт для навігації автономного агента на прикладі симульованого безпілотного літального апарата.

Об’єктом дослідження є процес створення ігрових карт та ландшафтів у віртуальних середовищах.

Предметом дослідження є алгоритми процедурної генерації ігрових карт і ландшафтів та їх властивості.

Методи дослідження, використані в роботі, включають аналіз наукових джерел, алгоритмічне моделювання, комп’ютерні експерименти, а також порівняльний

аналіз різних підходів до процедурної генерації. Для візуалізації результатів застосовуються засоби комп’ютерної графіки та симуляційні середовища.

Практичне значення отриманих результатів полягає у можливості використання розроблених алгоритмів процедурної генерації під час створення ігрових рівнів, інтерактивних симуляцій і тренувальних середовищ для автономних агентів. Запропонований підхід може бути використаний як у розробці комп’ютерних ігор, так і в дослідницьких або навчальних проєктах, пов’язаних з моделюванням поведінки агентів у складних просторових середовищах.

Дипломна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі розглядаються теоретичні основи процедурної генерації ігрових карт та ландшафтів. У другому розділі виконується аналіз алгоритмів процедурної генерації з точки зору структури ігрових середовищ та поведінки автономних агентів. Третій розділ присвячений розробці та інтерпретації власної системи процедурної генерації ігрових карт, а також демонстрації можливості її застосування на прикладі автономного агента.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ІГРОВИХ КАРТ ТА ЛАНДШАФТІВ

1.1. Вступ до процедурної генерації як галузі дослідження

Процедурна генерація на сьогодні є однією з ключових технологій у сфері комп’ютерних ігор, симуляцій та віртуальних середовищ. Вона дозволяє створювати великі, різноманітні, складні світи без необхідності ручного моделювання кожного елементу. Хоча первинно такі методи застосовувались переважно з метою економії часу та ресурсів розробників, нині процедурна генерація стала інструментом, що формує нові типи ігрового досвіду, створює невичерпну варіативність середовищ і забезпечує адаптивність до дій користувача або агента.

У класичній комп’ютерній графіці ключова мета — описати об’єкти та простори за допомогою детальних моделей. Проте у випадках, коли сцена має змінюватися динамічно, або коли необхідно створювати сотні чи тисячі унікальних комбінацій, ручне моделювання стає неможливим. Саме тоді виникає потреба в алгоритмах, здатних автоматично виробляти структури, рельєфи, об’єкти та їх взаємодію. Процедурна генерація відповідає на цю потребу, комбінуючи математичні моделі, теорію випадковості і формальні правила.

Найважливішою відмінністю процедурної генерації є наявність процесу, а не лише результату. Будь-який процедурно згенерований об’єкт — це наслідок програмно описаної послідовності операцій, які можна повторити з іншими вхідними параметрами для отримання нових варіантів. Завдяки цьому розробника цікавить не конкретна карта чи рівень, а алгоритм, який породжує нескінченну множину таких карт.

Таким чином, процедурна генерація постає як область, що поєднує графіку, моделювання, штучний інтелект, теорію алгоритмів і чисельні методи. Її значення постійно зростає, особливо у сфері симуляцій і навчання автономних

систем, де вимога до різноманітності середовища набагато вища, ніж у традиційних ігрових застосуваннях.

Історично процедурна генерація виникла як відповідь на жорсткі обмеження обчислювальних ресурсів і пам’яті ранніх комп’ютерних систем. Одними з перших прикладів її застосування стали ігри Elite та Rogue(рис. 1.1), у яких процедурні алгоритми використовувалися для створення великих ігрових просторів за мінімального обсягу збережених даних. У цих проєктах процедурна генерація виконувала не лише технічну функцію, але й формувала унікальний ігровий досвід, де кожне проходження відрізнялося від попереднього.

Рисунок 1.1 – Процедурна мапа Rogue

З розвитком апаратного забезпечення роль процедурної генерації поступово трансформувалася. Якщо раніше вона застосовувалася переважно з метою економії ресурсів, то згодом стала інструментом підвищення реіграбельності, варіативності та складності ігрових середовищ. У сучасних іграх процедурна генерація дедалі частіше використовується як засіб створення умов для emergent gameplay — ситуацій, що виникають спонтанно внаслідок взаємодії гравця з динамічним середовищем.

1.2. Поняття карти та ландшафту в контексті комп’ютерних симуляцій

Під терміном карта у комп’ютерних іграх і симуляторах зазвичай розуміють абстрактне або просторове представлення середовища, в якому знаходиться ігровий агент або користувач. Це може бути двовимірне поле, тривимірна сцена

або багаторівнева структура. Карта включає як геометрію простору (висоту, глибину, межі), так і функціональні елементи: маршрути руху, перешкоди, укриття, об’єкти взаємодії.

Ландшафт є спеціальним випадком карти — він стосується насамперед природних форм рельєфу, таких як гори, пагорби, долини, рівнини, скелі або русла річок. На відміну від абстрактних рівнів, ландшафт завжди має властивості, близькі до реальних умов. Він визначає:

- складність пересування;
- видимість і огляд;
- можливі маршрути;
- варіанти тактичної поведінки;
- динаміку навігаційних алгоритмів.

Чим реалістичніший ландшафт, тим складніші форми поведінки може виробити агент. Саме тому природні ландшафти часто застосовуються як основа тренувальних симуляторів — вони дають змогу тестувати алгоритми у багатоваріантних просторах.

Карти та ландшафти також можуть бути:

- детермінованими (завжди однаковими),
- псевдовипадковими (визначаються seed параметром),
- динамічними (змінюються в реальному часі).

Процедурна генерація здатна охопити всі ці варіанти.

1.3. Основні принципи процедурної генерації

Будь-яка процедура генерації базується на трьох фундаментальних принципах:

- Принцип повторюваності:

Важливим принципом процедурної генерації є повторюваність результатів за фіксованих початкових параметрів. Використання початкового зерна випадковості (seed) дозволяє відтворювати однакові ігрові карти або

ландшафти, що є критично важливим для тестування, налагодження та порівняльного аналізу алгоритмів.

- Принцип масштабованості:

Процедурні алгоритми повинні забезпечувати масштабованість, тобто можливість створення середовищ різного розміру без суттєвої зміни логіки генерації. Масштабованість дозволяє використовувати один і той самий алгоритм як для невеликих рівнів, так і для великих відкритих світів.

- Принцип валідації:

Автоматична генерація потребує механізмів перевірки коректності результатів. До таких механізмів належать перевірка зв’язності простору, прохідності ключових зон та відповідності карти заданим обмеженням. Без етапу валідації процедурна генерація може призводити до створення непридатних для гри середовищ.

1.3.1. Стохастичність з контрольованою випадковістю

Алгоритм має створювати різні результати при різних значеннях seed, але водночас зберігати:

- коректність структури;
- виконання правил;
- відповідність обмеженням.

Випадковість задає різноманітність, але сама по собі не забезпечує якості. Тому більшість систем комбінує випадковість із правилами.

1.3.2. Ієрархічність формування структури

Більшість природних об’єктів формуються багаторівнево. Наприклад:

1. спочатку з’являється базовий рельєф;
2. потім — великі форми;
3. далі — деталі;
4. на завершення — дрібні нерівності та шум.

Процедурні алгоритми відтворюють цю ієрархію. Зокрема, застосування фрактальних шумів або поетапних графових методів дає форму, подібну до реальних природних процесів.

1.3.3. Правила поведінки та обмеження

І навіть випадковий процес має підпорядковуватись певним законам. Для природних ландшафтів такими законами є:

- гравітація;
- гідродинаміка;
- осадові процеси;
- ерозія.

Алгоритми відтворюють їх приблизно, але цього достатньо, аби отримати правдоподібні форми.

1.4. Математичні моделі та підходи, які лежать в основі процедурної генерації

У теоретичному плані процедурна генерація спирається на кілька ключових математичних основ.

Математичні моделі, що використовуються в процедурній генерації, мають різні властивості та обмеження. Наприклад, шум Перліна забезпечує плавні переходи та добре підходить для створення природних ландшафтів, однак при некоректних параметрах може породжувати однотипні структури. Альтернативні підходи, такі як Simplex Noise або Value Noise, частково усувають ці недоліки, але мають власні обмеження з точки зору обчислювальної складності.

Клітинні автомати, на відміну від шумових методів, формують структуру середовища на основі локальних правил взаємодії клітин. Це дозволяє створювати більш виражені просторові структури, зокрема печери або замкнені зони. Проте використання лише клітинних автоматів без початкової випадкової ініціалізації може призводити до надмірної регулярності карти.

1.4.1. Стохастичні поля і шумові функції

Шумові функції — одна з найпродуктивніших моделей для створення плавних, неперіодичних структур. Між простим випадковим шумом та шумом Перліна існує принципова різниця(рис. 1.2): випадковий шум не має кореляції між сусідніми вузлами, тому виглядає «зернисто». Натомість Перлін ввів концепцію гладкої інтерполяції градієнтів, що створює цілісні контури, придатні для рельєфу.

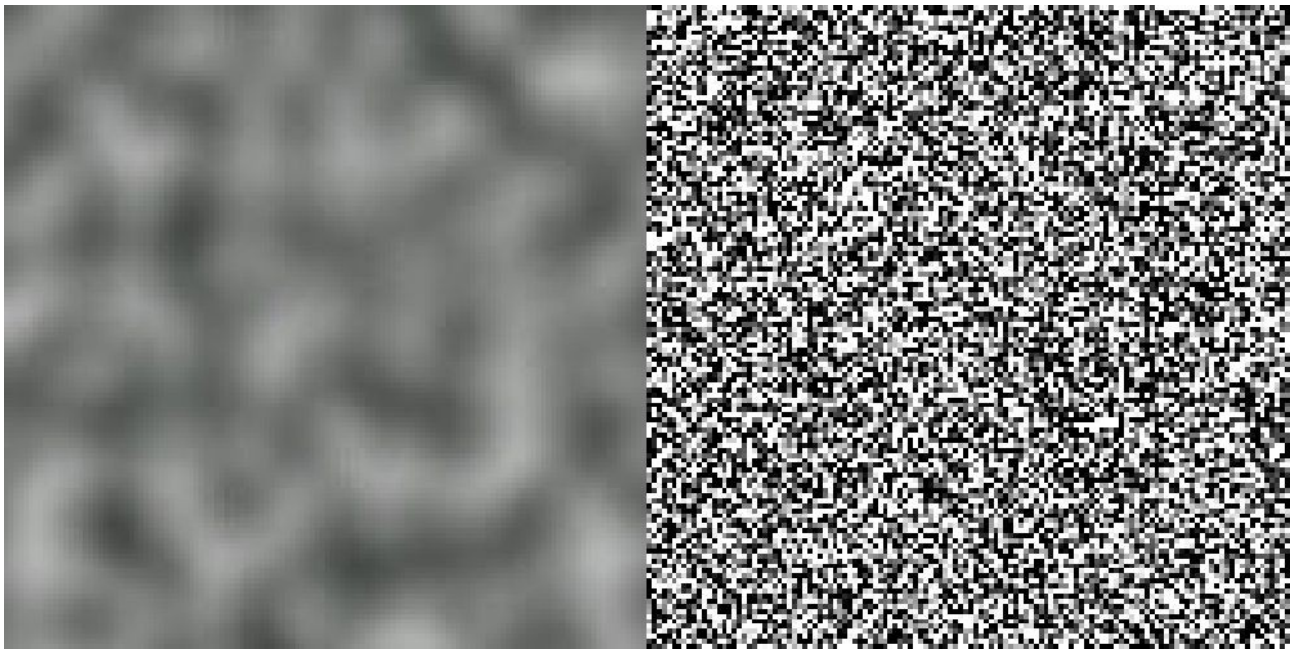

Рисунок 1.2 – Шум Перліна і випадковий шум.

Подальші модифікації, такі як Simplex Noise, OpenSimplex та Fractal Brownian Motion(рис. 1.3), розширили спектр можливих форм. Вони дозволяють породжувати:

- великі масиви гір;
- дрібні нерівності;
- мультифрактальні структури;
- варіативні текстури.

Комбінування шумових функцій із різними частотами формує реалістичні природні патерни.

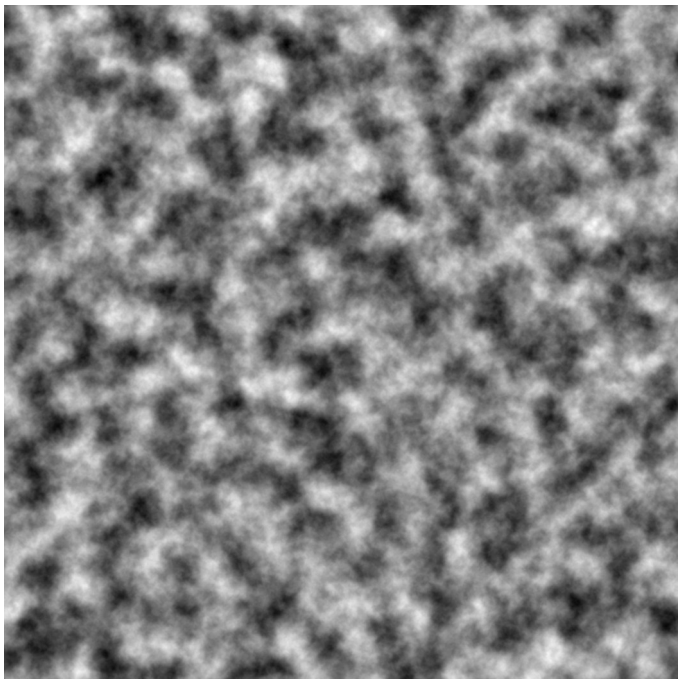

Рисунок 1.3 – Fractal Brownian Motion.

1.4.2. Графові структури та топологія простору

Граф — це універсальна модель зв’язків між точками. У процедурній генерації він дозволяє:

- формувати дороги;
- будувати структуру підземель;
- визначати маршрути;
- забезпечувати досяжність різних зон карти.

Рисунок 1.4 – L-systems graphs.

Методи генерації графів (MST(рис. 1.5), random walk, L-systems graph(рис. 1.4), Voronoi, Delaunay) забезпечують формування природних або функціональних структур.

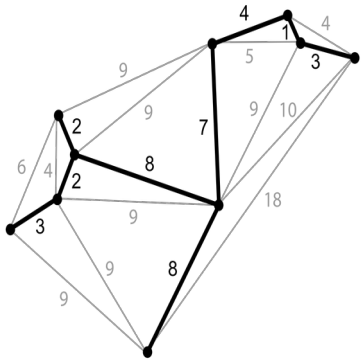

Рисунок 1.5 – MST(minimum spanning tree).

1.4.3. Клітинні автомати як модель локальної взаємодії

Клітинні автомати моделюють системи, де кожний елемент змінюється залежно від локальних правил. У процедурній генерації вони застосовуються для:

- формування печер;
- генерації органічних форм;
- імітації руйнування;
- проростання структур;
- моделювання рідин.

Перевага клітинних автоматів — природна непередбачуваність при збереженні внутрішньої логіки. На рисунку 1.6 можна побачити вплив клітинної автомати.

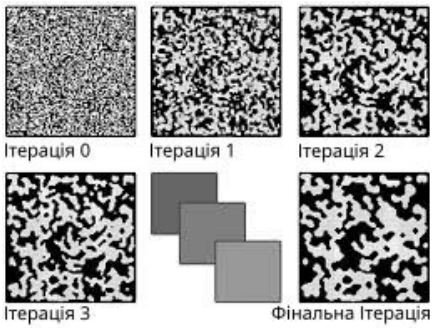

Рисунок 1.6 – Ітерації процедурної генерації з використанням клітинної автомати.

1.4.4. Фізичні симуляції та ерозійні процеси

Ерозія є ключовим фактором у природному моделюванні ландшафтів. Алгоритми ерозії наближають поведінку:

- водних потоків;
- зсувів ґрунту;
- вітрового впливу.

Навіть груба симуляція ерозії здатна значно покращити правдоподібність ландшафту.

1.5. Типи процедурної генерації та їх властивості

Процедурну генерацію умовно поділяють на кілька типів.

1.5.1. Детермінована генерація

Створює завжди однаковий результат при однакових параметрах. Зручна для відтворення рівнів без використання файлів.

1.5.2. Псевдовипадкова генерація

Залежить від seed. Використовується найчастіше, оскільки поєднує повторюваність і різноманітність.

1.5.3. Динамічна або адаптивна генерація

Може змінювати карту під час ігрового процесу або в залежності від поведінки агента. Це найскладніший тип, але він дозволяє створювати симуляційні середовища, що еволюціонують.

1.6. Процедурна генерація в ігровій індустрії: стан і тенденції

У сучасних проєктах процедурна генерація стала стандартним інструментом. Відомі приклади:

- Minecraft — нескінченні світи на основі шумів і багаторівневих структур.
- No Man’s Sky(рис. 1.7) — мультискладні фрактальні моделі для планетарних систем.
- Hades і Spelunky — динамічні рівні, що змінюють структуру підземель.

- Стратегічні ігри — карти, що кожен раз створюються наново.

Рисунок 1.7 – Процедурна генерація No Man’s Sky.

Головною тенденцією є зближення симуляцій і процедурної генерації. Алгоритми вже не просто створюють геометрію — вони формують цілі світи з екосистемами, логікою поведінки та змінними параметрами.

1.7. Значення процедурної генерації для робототехніки та симуляцій

У галузі навчання автономних систем (дронів, роботів, безпілотних автомобілів) виникає потреба у величезній кількості варіативних середовищ. Це необхідно для:

- перевірки стійкості алгоритмів;
- зменшення перенавчання;
- підготовки до непередбачуваних ситуацій;
- прискорення розвитку RL-агентів.

Статичні карти швидко вичерпують себе: агенти запам’ятовують їх структуру, перестаючи реагувати на нові сценарії. Процедурно згенеровані середовища розв’язують цю проблему, створюючи синтетичне «багатосвіття», у якому можливі тисячі різних конфігурацій.

Оскільки реальне середовище не повторюється, а подекуди змінюється випадково, моделі повинні отримувати досвід роботи в умовах, максимально наближених до природних. Саме тому процедурна генерація стає основою сучасних платформ для автономних агентів.

Окрім ігрових застосувань, процедурна генерація широко використовується у симуляційних середовищах для навчання та тестування автономних агентів. Однією з ключових концепцій у цьому контексті є domain randomization, яка полягає у варіюванні параметрів середовища з метою підвищення узагальнювальної здатності агентів.

Процедурно згенеровані середовища дозволяють створювати велику кількість сценаріїв взаємодії без ручного втручання, що значно прискорює процес експериментів. Такий підхід особливо ефективний у випадках, коли необхідно моделювати поведінку агентів у складних або непередбачуваних умовах.

1.8. Обмеження та проблеми процедурної генерації ігрових карт та ландшафтів

Незважаючи на значні переваги процедурної генерації ігрових карт та ландшафтів, застосування даних методів супроводжується низкою обмежень і проблем, які необхідно враховувати під час проєктування ігрових середовищ. Неконтрольоване використання алгоритмів генерації може негативно впливати на геймплей, баланс гри та загальне враження користувача.

Однією з основних проблем процедурної генерації є втрата дизайнерського контролю. На відміну від ручного створення рівнів, де кожен елемент ретельно продумується розробником, процедурні алгоритми працюють на основі формальних правил і випадкових величин. Це може призводити до появи ігрових карт, які є технічно коректними, але нецікавими або непридатними з точки зору ігрового процесу. Наприклад, згенеровані простори можуть бути надто порожніми, монотонними або, навпаки, перенасиченими перешкодами.

Іншою суттєвою проблемою є баланс складності ігрових карт. Процедурно згенеровані рівні можуть суттєво відрізнятися за складністю навіть при однакових початкових параметрах. Це створює труднощі в забезпеченні стабільного ігрового досвіду, оскільки користувач може зіткнутися з надто складним або надто простим рівнем без очевидних на те причин. Особливо критичною ця проблема є для ігор, орієнтованих на змагальний або прогресивний геймплей.

Проблема зв’язності та прохідності ігрового простору також є однією з ключових при процедурній генерації. Без додаткових перевірок і корекцій алгоритм може створювати ізольовані області, тупикові маршрути або ситуації, в яких ключові ігрові зони стають недоступними. Це вимагає застосування додаткових механізмів аналізу графа карти, перевірки досяжності та автоматичного виправлення помилок генерації.

Ще одним обмеженням є складність створення осмисленого контенту. Процедурна генерація добре підходить для формування геометрії простору, але значно гірше справляється зі створенням сюжетно навантажених або емоційно значущих елементів. Через це процедурні рівні часто потребують поєднання з ручним дизайном або сценарними елементами, що зменшує рівень автоматизації, але підвищує якість кінцевого результату.

Важливою проблемою є також повторюваність шаблонів. Навіть при використанні випадкових чисел багато алгоритмів процедурної генерації з часом починають створювати подібні за структурою карти. Досвідчений гравець може швидко розпізнати ці патерни, що знижує ефект новизни та реіграбельність гри. Для зменшення цього ефекту необхідно використовувати складніші комбіновані алгоритми або механізми адаптації параметрів генерації.

Окрему увагу слід приділити обчислювальним обмеженням. Деякі алгоритми процедурної генерації, особливо ті, що включають багатокрокову оптимізацію або симуляцію фізичних процесів (наприклад, ерозії), можуть вимагати значних

обчислювальних ресурсів. Це обмежує можливість їх використання в режимі реального часу або на платформах з обмеженою продуктивністю.

Застосування процедурної генерації в симуляційних середовищах з автономними агентами додає ще один рівень складності. Згенероване середовище повинно бути не лише візуально правдоподібним, але й придатним для навігації та прийняття рішень агентом. Невдало згенерована структура простору може ускладнити або повністю заблокувати виконання агентом поставлених завдань, що потребує додаткової валідації результатів генерації.

Таким чином, процедурна генерація ігрових карт та ландшафтів є потужним інструментом, але її ефективне застосування можливе лише за умови поєднання алгоритмічних методів із дизайнерськими обмеженнями та механізмами контролю якості. Усвідомлення та врахування наведених проблем дозволяє значно підвищити практичну цінність процедурно згенерованих ігрових середовищ.

РОЗДІЛ 2 АНАЛІЗ АЛГОРИТМІВ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ІГРОВИХ КАРТ ТА ЇХ ВПЛИВ НА ГЕЙМПЛЕЙ І ПОВЕДІНКУ АВТОНОМНИХ АГЕНТІВ

2.1. Ігрова карта як основа формування геймплею

У сучасних комп’ютерних іграх карта або рівень перестає бути лише фоном для подій. Вона безпосередньо визначає характер ігрового процесу, впливає на темп гри, складність, стиль пересування гравця та поведінку автономних ігрових агентів. Розташування перешкод, відкритих просторів, вузьких проходів і зон видимості формує сценарії взаємодії, які можуть бути як заздалегідь передбаченими, так і такими, що виникають спонтанно.

Процедурна генерація змінює підхід до проєктування ігрових карт. Якщо в традиційному дизайні рівнів кожен маршрут ретельно продумується, то у процедурних системах дизайнер фактично проєктує правила створення простору, а не сам простір. Це означає, що ключовим об’єктом аналізу стає не окрема карта, а клас карт, які можуть бути згенеровані заданим алгоритмом.

З точки зору геймплею важливими є такі властивості карти, як зв’язність простору, наявність альтернативних маршрутів, баланс між відкритими та замкненими зонами, а також можливість створення ситуацій ризику або стратегічного вибору. Саме ці характеристики визначають, чи буде ігровий процес динамічним, напруженим або передбачуваним.

Ігрова карта виконує роль не лише просторового контейнера для ігрових об’єктів, але й активного елементу геймплею, що формує набір можливих дій гравця. У теорії ігрового дизайну такі можливості часто описуються терміном affordances, який визначає, які дії середовище «пропонує» користувачу. Процедурна генерація безпосередньо впливає на формування цих можливостей, оскільки змінює структуру простору, розташування проходів і зон взаємодії.

Зміна просторових характеристик карти може суттєво впливати на темп гри, рівень напруження та стиль проходження. Наприклад, наявність вузьких

коридорів створює умови для обережної гри, тоді як відкриті простори стимулюють дослідження та агресивні стратегії. Таким чином, алгоритми процедурної генерації виступають непрямим інструментом керування геймплейними сценаріями.

2.2. Навігація в процедурно згенерованих середовищах

Навігація є однією з базових механік більшості ігор. Вона стосується не лише переміщення гравця, але й поведінки автономних агентів, таких як вороги, союзники або нейтральні персонажі. У процедурно згенерованих картах навігація набуває особливої складності, оскільки середовище не є фіксованим і може змінюватися від запуску до запуску.

Для автономних агентів навігація включає декілька рівнів:

- локальне уникнення перешкод;
- пошук маршруту між зонами;
- адаптацію до змін структури карти;
- ухвалення рішень у випадку кількох можливих шляхів.

Алгоритми процедурної генерації по-різному впливають на ці рівні. Наприклад, карти з плавним рельєфом і великою кількістю відкритого простору створюють умови для простих стратегій руху, тоді як карти з великою кількістю вузьких проходів змушують агентів часто перебудовувати маршрут і реагувати на локальні зміни.

З ігрової точки зору це означає, що вибір алгоритму генерації безпосередньо визначає стиль поведінки NPC. У складних лабіринтоподібних середовищах агенти виглядають обережними і реактивними, тоді як у відкритих просторах — більш агресивними і прямолінійними.

Навігація в процедурно згенерованих ігрових середовищах має як локальний, так і глобальний характер. Локальна навігація пов’язана з уникненням безпосередніх перешкод і реагуванням на зміни середовища, тоді як глобальна навігація передбачає планування маршруту між віддаленими точками карти.

Процедурна генерація ускладнює глобальну навігацію, оскільки структура карти може істотно змінюватися при кожному запуску гри.

Для автономних агентів це означає необхідність адаптації алгоритмів пошуку шляху до різних топологічних конфігурацій. У середовищах з великою кількістю розгалужень алгоритми навігації мають працювати з більшим простором станів, що безпосередньо впливає на обчислювальну складність і стабільність поведінки агента.

2.3. Вплив процедурної генерації на складність і реіграбельність гри

Однією з головних переваг процедурної генерації є підвищення реіграбельності. Гравець не може запам’ятати карту, оскільки вона змінюється щоразу. Це створює відчуття новизни та непередбачуваності, що особливо цінується в жанрах roguelike, survival та sandbox.

Проте зростання варіативності водночас ускладнює контроль складності. Якщо алгоритм генерації не враховує баланс, то деякі згенеровані карти можуть бути надто простими або, навпаки, практично непрохідними. Саме тому аналіз алгоритмів процедурної генерації повинен враховувати не лише естетичні характеристики, але й їх вплив на ігровий процес.

У цьому контексті важливою є концепція керованої випадковості, коли випадкові елементи обмежуються певними правилами. Наприклад, кількість розгалужень, довжина коридорів або щільність перешкод можуть змінюватися в заданих межах. Такий підхід дозволяє зберегти баланс між несподіваністю та прохідністю рівнів.

Однією з ключових задач процедурної генерації є забезпечення збалансованої складності ігрових карт. На відміну від статичних рівнів, де складність може бути ретельно налаштована вручну, процедурні середовища потребують використання формальних обмежень та евристик. До таких обмежень належать мінімальна кількість альтернативних маршрутів, контрольована щільність перешкод і обмеження на довжину шляхів.

Реіграбельність гри безпосередньо пов’язана з варіативністю карт, проте надмірна випадковість може призводити до несправедливих або непрохідних сценаріїв. Тому сучасні системи процедурної генерації все частіше використовують підхід керованої випадковості, який дозволяє зберігати баланс між новизною та передбачуваністю ігрового процесу.

2.4. Аналіз алгоритмів генерації з погляду структури ігрового простору

Різні алгоритми процедурної генерації створюють принципово різні типи ігрових просторів. Їх аналіз доцільно проводити не за формальними характеристиками, а за тим, які ігрові ситуації вони породжують.

Аналіз алгоритмів процедурної генерації доцільно проводити не лише за технічними характеристиками, але й з позиції ігрового досвіду. Алгоритми, що створюють плавні та відкриті ландшафти, зазвичай сприяють спокійному дослідницькому геймплею, тоді як алгоритми, орієнтовані на формування лабіринтоподібних структур, створюють напружені та динамічні ігрові ситуації.

З погляду автономних агентів різні алгоритми генерації формують різні умови навігації та прийняття рішень. Це підтверджує необхідність комбінування підходів, оскільки жоден окремий алгоритм не здатний забезпечити оптимальний баланс між різноманітністю та ігровою доцільністю.

2.4.1. Шумові алгоритми та відкриті ігрові простори

Алгоритми, засновані на шумових функціях, найчастіше використовуються для створення природних ландшафтів. Вони формують плавні перепади висот і великі відкриті простори, що добре підходять для ігор з дослідженням світу або вільним пересуванням.

З точки зору геймплею такі середовища зазвичай:

- сприяють дальньому огляду;
- дозволяють будувати довгі траєкторії руху;
- зменшують кількість несподіваних зіткнень.

Для автономних агентів це означає спрощену навігацію, але водночас меншу кількість тактичних рішень. У таких умовах поведінка агентів стає більш передбачуваною.

2.4.2. Клітинні автомати та замкнені структури

Клітинні автомати часто застосовуються для створення печер, підземель або зруйнованих просторів. Вони генерують середовища з великою кількістю вузьких проходів, нерівних стін і замкнених зон.

Такі карти створюють напружений геймплей, оскільки:

- обмежують огляд;
- ускладнюють навігацію;
- змушують гравця і агентів часто змінювати напрям руху.

Для автономних агентів це є складним сценарієм, який потребує ефективного локального уникнення перешкод і швидкого реагування на зміну оточення.

2.4.3. Графові методи та структуровані рівні

Графові підходи дозволяють створювати ігрові карти з чіткою логічною структурою. Такі рівні часто мають центральні вузли, ключові зони та альтернативні маршрути між ними.

У ігровому сенсі це сприяє:

- стратегічному плануванню;
- контролю потоків руху;
- створенню зон підвищеного ризику або нагороди.

Автономні агенти в таких середовищах можуть демонструвати складні форми поведінки, зокрема патрулювання, засідки або переслідування.

2.5. Комбінування алгоритмів як засіб підвищення якості ігрових карт

Аналіз показує, що використання одного типу алгоритму рідко дає оптимальний результат. Найбільш цікаві ігрові середовища виникають при комбінуванні різних підходів. Наприклад, шумовий ландшафт може слугувати

основою, на якій розміщуються структуровані елементи, створені графовими методами, а клітинні автомати додають локальні ускладнення.

Такий підхід дозволяє:

- поєднувати відкриті та замкнені простори;
- створювати різні сценарії руху;
- формувати непередбачувані ігрові ситуації.

Комбіновані алгоритми є особливо корисними у випадках, коли карта повинна підтримувати як вільне дослідження, так і інтенсивні тактичні взаємодії.

2.6. Поведінка автономних агентів у процедурно згенерованих ігрових середовищах

У процедурно згенерованих картах автономні агенти не можуть покладатися на заздалегідь відомі маршрути. Вони змушені аналізувати середовище в реальному часі, що робить їхню поведінку більш природною і менш шаблонною.

У якості автономного агента може виступати будь-який ігровий об’єкт, здатний самостійно переміщуватися та ухвалювати рішення. У межах цієї роботи як експериментальний приклад розглядається також безпілотний літальний апарат у симуляції, що дозволяє дослідити навігацію в тривимірному просторі. Проте принципи поведінки такого агента є універсальними і застосовні до широкого спектра ігрових персонажів.

Процедурно згенеровані карти суттєво впливають на поведінкову варіативність автономних агентів. Зміна структури простору призводить до зміни набору доступних дій, що змушує агентів використовувати різні стратегії навігації та взаємодії з середовищем. У таких умовах поведінка агента стає менш шаблонною і більш наближеною до поведінки реального гравця.

Для систем зі штучним інтелектом це означає розширення простору станів та підвищення складності задачі прийняття рішень. Процедурна генерація, таким

чином, виступає не лише інструментом створення контенту, але й засобом підвищення складності та реалістичності поведінки агентів.

2.7. Вплив структури ігрової карти на геймплей та стратегії навігації

Структура ігрової карти є одним із ключових факторів, що визначає характер геймплею та поведінку як гравця, так і автономних агентів. Навіть за однакових ігрових механік зміна просторової організації середовища може суттєво впливати на складність проходження, стиль гри та прийняття тактичних рішень. У контексті процедурної генерації ця залежність набуває особливої ваги, оскільки структура карти формується автоматично та може змінюватися при кожному запуску гри.

Однією з базових характеристик ігрової карти є ступінь відкритості простору. Відкриті карти з великою кількістю альтернативних маршрутів стимулюють дослідницький стиль гри та дозволяють гравцю обирати власну стратегію пересування. У таких середовищах навігація зазвичай менш лінійна, а ризик і винагорода розподіляються рівномірніше. Для автономних агентів відкриті простори полегшують пошук маршруту, але водночас збільшують кількість можливих варіантів рішень, що ускладнює оптимізацію поведінки.

На противагу цьому, лабіринтоподібні та замкнені структури формують більш контрольований і напружений геймплей. Обмежена кількість проходів змушує гравця ретельно планувати рух, враховувати розташування перешкод і можливі точки небезпеки. Для автономних агентів такі карти створюють складні навігаційні задачі, оскільки помилки у виборі маршруту можуть призводити до значних затримок або повного блокування руху.

Важливу роль відіграє топологія карти, тобто характер зв’язків між окремими зонами. Карти з деревоподібною структурою мають чітко визначений шлях проходження, що спрощує орієнтацію, але зменшує варіативність геймплею. Натомість карти з циклічними або графовими структурами створюють умови для альтернативних маршрутів, обхідних шляхів і тактичного маневрування. У

процедурній генерації такі структури часто використовуються для підвищення реіграбельності.

Ще одним важливим аспектом є розподіл зон інтересу та ключових точок. Якщо такі зони розміщені нерівномірно або випадково, це може призводити до дисбалансу ігрового процесу, коли певні ділянки карти стають надто важливими, а інші — майже не використовуються. Процедурні алгоритми повинні враховувати не лише геометрію простору, але й логіку розміщення ігрових об’єктів, що впливають на поведінку гравця.

Структура карти також безпосередньо впливає на динаміку складності. У процедурно згенерованих середовищах складність може змінюватися не за сценарієм, а залежно від конкретної конфігурації простору. Наприклад, висока щільність перешкод або вузькі проходи можуть суттєво ускладнювати проходження, навіть якщо ігрові механіки залишаються незмінними. Це створює додаткові виклики для балансування гри.

З точки зору автономних агентів, структура карти визначає ефективність алгоритмів навігації та прийняття рішень. У простих і відкритих середовищах агенти можуть використовувати базові алгоритми пошуку шляху, тоді як у складних багаторівневих структурах виникає потреба у ієрархічних або адаптивних методах навігації. Таким чином, якість процедурної генерації безпосередньо впливає на стабільність та передбачуваність поведінки агентів.

У контексті ігрового дизайну важливим є поняття emergent gameplay, коли складні ігрові ситуації виникають як наслідок взаємодії гравця з процедурно згенерованою структурою карти. Різноманітність просторових конфігурацій створює унікальні сценарії проходження, які неможливо передбачити заздалегідь. Саме цей ефект є однією з головних переваг процедурної генерації в сучасних іграх.

Отже, структура ігрової карти відіграє визначальну роль у формуванні геймплею, стратегій навігації та поведінки автономних агентів. Процедурна

генерація дозволяє створювати різноманітні просторові конфігурації, проте вимагає ретельного аналізу та контролю для забезпечення збалансованого ігрового досвіду.

Однією з ключових переваг процедурної генерації є створення умов для emergent gameplay — ігрових ситуацій, що виникають спонтанно внаслідок взаємодії гравця або автономного агента зі складним середовищем. Різноманітність просторових конфігурацій призводить до появи унікальних сценаріїв проходження, які неможливо повністю передбачити на етапі проєктування алгоритму.

Такий підхід значно підвищує цінність процедурної генерації для сучасних ігор, оскільки дозволяє створювати неповторний ігровий досвід без ручного створення кожного рівня.

РОЗДІЛ 3 РОЗРОБКА ТА ІНТЕРПРЕТАЦІЯ СИСТЕМИ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ІГРОВИХ КАРТ

3.1. Постановка задачі розробки ігрового середовища

На основі теоретичних положень та аналітичних висновків, отриманих у попередніх розділах, було сформульовано задачу розробки системи процедурної генерації ігрових карт. Основною вимогою до системи є здатність автоматично створювати різноманітні ігрові середовища зі збереженням керованої складності та логічної структури простору.

Система повинна забезпечувати:

- варіативність ігрових карт;
- керованість параметрів генерації;
- підтримку навігації автономних агентів;
- можливість візуалізації результатів генерації.

Ігрова карта розглядається як дво- або тривимірне середовище, що складається з прохідних та непрохідних зон, а також зон зі спеціальними властивостями. При цьому генерація карти не прив’язується до конкретного жанру гри, що забезпечує універсальність розробленого підходу.

Реалізована система процедурної генерації ігрових карт побудована як послідовний конвеєр обробки даних, у якому кожен етап виконує чітко визначену функцію. Такий підхід дозволяє ізолювати окремі компоненти системи, спростити тестування та забезпечити можливість подальшого розширення функціональності без суттєвих змін у базовій архітектурі.

Концепція системи орієнтована на практичне використання в ігрових застосунках, де важливими є не лише візуальні характеристики карт, але й їх навігаційна придатність, стабільність генерації та передбачуваність поведінки автономних агентів.

3.2. Архітектура системи процедурної генерації

Розроблена система має модульну архітектуру, що дозволяє легко змінювати або доповнювати окремі компоненти. Загальна структура системи включає такі основні модулі:

- модуль генерації базового ландшафту;
- модуль структурної обробки карти;
- модуль перевірки зв’язності та прохідності;
- модуль розміщення агентів;
- модуль візуалізації.

Модуль генерації базового ландшафту відповідає за створення первинної геометрії карти. На цьому етапі формується загальна структура простору без урахування ігрових об’єктів. Далі отриманий ландшафт передається до модуля структурної обробки, який додає логічні елементи, такі як проходи, зони інтересу або обмеження руху.

Перевірка зв’язності є критично важливим етапом, оскільки вона гарантує можливість навігації по карті. Система автоматично аналізує, чи існують прохідні маршрути між ключовими зонами, і за потреби коригує структуру карти.

Архітектура системи побудована за модульним принципом, що дозволяє розділити функціональність на логічно незалежні блоки. Модуль генерації відповідає за створення базової структури карти, модуль обробки — за покращення та згладжування результату, а модуль валідації — за перевірку прохідності та зв’язності простору.

Такий розподіл обов’язків зменшує зв’язаність компонентів та полегшує заміну або модифікацію окремих алгоритмів. Наприклад, клітинний автомат може бути замінений іншим методом обробки без необхідності змінювати логіку валідації або навігації.

3.3. Реалізація алгоритму комбінованої генерації

З урахуванням результатів аналізу, викладених у другому розділі, у системі було реалізовано комбінований підхід до процедурної генерації. Його сутність полягає у поєднанні різних алгоритмічних методів для досягнення балансу між природністю ландшафту та структурованістю ігрового простору.

На першому етапі формується базовий ландшафт, який задає загальні характеристики середовища. Далі застосовуються алгоритми локальної модифікації, що створюють зони підвищеної складності або навігаційні вузли. Завершальним етапом є корекція результатів генерації відповідно до заданих обмежень, таких як мінімальна зв’язність або допустима кількість перешкод.

Такий підхід дозволяє уникнути крайнощів, характерних для використання лише одного типу алгоритму, та забезпечує створення ігрових карт, які є водночас варіативними і прохідними.

Застосування комбінованого підходу до процедурної генерації дозволяє поєднати переваги різних алгоритмів та мінімізувати їх недоліки. Початкова генерація на основі шуму забезпечує випадковість і різноманітність, тоді як клітинний автомат виконує роль локального оптимізатора, що формує більш природні структури.

Подальший етап валідації гарантує відповідність карти базовим вимогам ігрового процесу, зокрема зв’язність простору та можливість навігації. Таким чином, система не обмежується генерацією випадкового контенту, а формує ігрове середовище з урахуванням функціональних вимог.

3.4. Параметризація та керування складністю

Однією з ключових вимог до системи є можливість керування складністю згенерованих карт. Для цього було введено набір параметрів, які впливають на результат генерації, зокрема:

- щільність перешкод;
- рівень розгалуженості шляхів;

- співвідношення відкритих і замкнених просторів;
- кількість зон інтересу.

Зміна цих параметрів дозволяє адаптувати ігрове середовище під різні сценарії використання — від дослідницьких ігор до динамічних екшенів. Важливо, що параметризація не порушує цілісність карти, оскільки всі параметри застосовуються в межах заздалегідь визначених допустимих значень.

3.5. Інтеграція автономного агента в ігрове середовище

Для демонстрації практичної придатності згенерованих ігрових карт було реалізовано сценарій взаємодії автономного агента з середовищем. Агент розглядається як узагальнений ігровий об’єкт, здатний до навігації та реагування на перешкоди.

У межах експериментального прикладу агент реалізований у вигляді симульованого безпілотного літального апарата, що переміщується у тривимірному просторі. Використання БПЛА дозволяє дослідити навігацію не лише в горизонтальній площині, але й з урахуванням висоти, що є характерним для сучасних ігрових середовищ.

Важливо підкреслити, що вибір БПЛА є умовним і не впливає на загальні принципи роботи системи. Розроблені ігрові карти можуть бути використані для будь-яких автономних агентів, зокрема ігрових персонажів або ботів.

3.6. Результати експериментів та їх інтерпретація

У процесі тестування системи було згенеровано низку ігрових карт з різними параметрами складності(рис. 3.1). Аналіз показав, що комбінований алгоритм генерації забезпечує стабільну зв’язність карт і створює різноманітні навігаційні сценарії.

Рисунок 3.1 – Приклад згенерованої ігрової карти в середовищі Unity

Автономний агент успішно адаптувався до різних структур середовища, що підтверджує універсальність розробленого підходу. У картах з високою щільністю перешкод спостерігалося збільшення часу навігації та кількості корекцій маршруту, тоді як у відкритих середовищах рух агента був більш прямолінійним.

Отримані результати свідчать про те, що запропонована система процедурної генерації є придатною для створення ігрових середовищ з різним рівнем складності та може бути використана як основа для подальших експериментів з поведінкою автономних агентів.

Результати експериментів підтверджують, що реалізована система здатна стабільно генерувати прохідні та зв’язні карти різних розмірів. Зміна параметрів генерації призводить до прогнозованих змін структури карти, що свідчить про керованість процесу.

Наявність автономного агента з алгоритмом пошуку шляху дозволяє не лише візуально оцінити результат генерації, але й перевірити його функціональну придатність у динаміці. Це є важливою перевагою системи з точки зору практичного застосування.

3.7. Обмеження реалізованої системи та напрями розвитку

Незважаючи на успішну реалізацію системи процедурної генерації ігрових карт, розроблена платформа має ряд обмежень, які визначають подальші можливості її вдосконалення та застосування. По-перше, поточна реалізація використовує фіксовані параметри алгоритмів генерації, що обмежує гнучкість створення нових типів карт. Зміни в конфігурації параметрів потребують ручного втручання або перезапуску системи, що ускладнює інтеграцію в динамічні ігрові середовища.

По-друге, система поки що не враховує адаптивну складність карт залежно від поведінки гравця або автономного агента. Це означає, що всі згенеровані карти мають однаковий рівень складності для всіх користувачів, що може призвести

до надмірної або недостатньої складності в окремих сценаріях. В майбутньому планується реалізація механізмів динамічної адаптації параметрів генерації, які змінюватимуть щільність перешкод, розгалуженість маршрутів та розташування зон інтересу на основі статистики ігрового процесу.

Третє обмеження стосується структурної різноманітності згенерованих карт. Хоча комбінований підхід до генерації забезпечує достатній рівень варіативності, у деяких випадках можуть виникати схожі або повторювані патерни, особливо при обмежених наборах правил та алгоритмічних елементів. Це зменшує ефект новизни і може вплинути на реіграбельність. Подальший розвиток системи передбачає використання більш складних комбінацій алгоритмів, включаючи фрактальні та еволюційні методи, які дозволяють створювати унікальні просторові конфігурації.

Четверте обмеження пов’язане з обчислювальними ресурсами. Деякі компоненти генерації, такі як фізично орієнтована ерозія ландшафту або багаторівнева корекція графових структур, потребують значних обчислень. Це може обмежувати застосування системи на платформах з низькою продуктивністю або при необхідності генерації карт у режимі реального часу. Одним із напрямів розвитку є оптимізація алгоритмів і застосування апаратного прискорення, зокрема через використання GPU та багатопоточних обчислень.

Ще одним напрямом покращення є інтеграція системи з механізмами штучного інтелекту. На даний момент автономний агент використовується лише для демонстрації придатності карт. В подальшому можливе застосування алгоритмів машинного навчання та reinforcement learning для адаптивної взаємодії агента зі структурою карт, оцінки складності та виявлення слабких зон ігрового середовища.

Крім того, наразі система не враховує емоційний та сюжетний контекст карт, що обмежує її застосування для жанрів, де важливу роль відіграють наратив та атмосферність. Розширення платформи шляхом введення алгоритмів, які

формують зони з конкретними ігровими подіями або сюжетними елементами, може значно підвищити її практичну цінність.

Таким чином, розроблена система демонструє ефективну роботу та дозволяє створювати варіативні та керовані за складністю ігрові карти, проте її функціональні можливості можна значно розширити. Основні напрями розвитку включають: динамічну адаптацію параметрів, оптимізацію обчислювальних алгоритмів, застосування більш складних генеративних методів, інтеграцію з адаптивними агентами та впровадження сюжетних або емоційних елементів у карту.

З урахуванням зазначених обмежень та потенційних напрямів розвитку, система є міцною основою для подальших досліджень у сфері процедурної генерації ігрового контенту та інтерактивних середовищ.

Реалізована система може бути використана як основа для створення ігрових застосунків жанру roguelike або для симуляційних середовищ з автономними агентами. Модульна архітектура дозволяє інтегрувати додаткові алгоритми генерації, а також розширювати систему за рахунок адаптивних механізмів складності.

Перспективними напрямами подальшого розвитку є інтеграція методів машинного навчання для адаптації параметрів генерації, використання більш складних моделей поведінки агентів та розширення системи сюжетними елементами.

Висновок

У даній дипломній роботі було виконано дослідження алгоритмів процедурної генерації ігрових карт та ландшафтів з акцентом на їх вплив на структуру ігрового середовища, геймплей і поведінку автономних агентів. Актуальність обраної теми зумовлена зростаючими вимогами до масштабності, варіативності та реіграбельності сучасних комп’ютерних ігор і симуляцій, а також необхідністю автоматизації процесу створення ігрового контенту.

У першому розділі роботи було розглянуто теоретичні основи процедурної генерації, проаналізовано ключові підходи до створення ігрових карт і ландшафтів, а також визначено основні переваги та обмеження процедурних методів порівняно з ручним проєктуванням рівнів. Показано, що процедурна генерація є ефективним інструментом для створення великих і різноманітних ігрових середовищ, здатних адаптуватися до потреб гравця або автономних агентів.

У другому розділі проведено аналіз алгоритмів процедурної генерації з точки зору їх впливу на геймплей, навігацію та поведінку автономних агентів. Було встановлено, що різні алгоритмічні підходи формують принципово різні типи ігрових просторів — від відкритих ландшафтів до замкнених лабіринтоподібних структур. Особливу увагу приділено концепції керованої випадковості та комбінуванню алгоритмів, що дозволяє досягти балансу між варіативністю, прохідністю та складністю ігрових карт.

У третьому розділі розроблено та описано власну систему процедурної генерації ігрових карт, яка базується на комбінованому алгоритмі. Запропонована архітектура системи забезпечує модульність, параметризацію та можливість керування складністю згенерованих середовищ. Для демонстрації практичної придатності системи було реалізовано сценарій взаємодії автономного агента з процедурно згенерованою картою, де агент представлено у вигляді симульованого безпілотного літального апарата.

Результати експериментів підтвердили, що розроблена система здатна створювати зв’язні, варіативні та ігрово доцільні карти, придатні як для використання в комп’ютерних іграх, так і в симуляційних середовищах для автономних агентів. Використання БПЛА в роботі має демонстраційний характер і підкреслює універсальність запропонованого підходу, який не обмежується конкретним типом агента або жанром гри.

Отримані результати можуть бути використані у подальших дослідженнях у сфері ігрового дизайну, процедурної генерації контенту, а також під час розробки тренувальних середовищ для автономних агентів. Перспективами подальшої роботи є розширення системи за рахунок адаптивної генерації, інтеграції методів машинного навчання та глибшого аналізу впливу структури карт на поведінку агентів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- Ebert D. Texturing and Modeling: A Procedural Approach. – Morgan Kaufmann, 2003.
- Shaker N., Togelius J., Nelson M. Procedural Content Generation in Games. – Springer, 2016.
- Togelius J., Yannakakis G. Artificial Intelligence and Games. – Springer, 2018.
- Millington I., Funge J. Artificial Intelligence for Games. – CRC Press, 2016.
- Adams E., Dormans J. Game Mechanics: Advanced Game Design. – New Riders, 2012.
- Johnson L., Yannakakis G. “PCG-based Game Design Patterns”. – IEEE Transactions on Games, 2019.
- Smelik R., Tutenel T., Bidarra R. “A Survey of Procedural Methods for Terrain Modelling”. – Computer Graphics and Applications, 2014.
- van der Linden R., Lopes R., Bidarra R. “Procedural Generation of Dungeons”. – IEEE Transactions on Computational Intelligence and AI in Games, 2013.
- Ashlock D. Evolutionary Computation for Modeling and Optimization. – Springer, 2006.
- Rabin S. Game AI Pro. – CRC Press, 2013.
- Yannakakis G., Togelius J. “Experience-Driven Procedural Content Generation”. – IEEE Transactions on Affective Computing, 2011.
- Gregory J. Game Engine Architecture. – CRC Press, 2018.
- Russell S., Norvig P. Artificial Intelligence: A Modern Approach. – Pearson, 2021.
- LaValle S. Planning Algorithms. – Cambridge University Press, 2006.
- Botea A., Müller M., Schaeffer J. “Near Optimal Hierarchical Path-Finding”. – Journal of Game Development, 2004.

Додаток А

Вихідний Код

MapGenerator.cs(Генерація мапи)

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

public class MapGenerator : MonoBehaviour
{
    [Header("Map Settings")]
    public int width = 50;
    public int height = 50;
    public int seed;

    [Header("Generation Settings")]
    public bool useRandomSeed = true;

    [Header("Cellular Automata")]
    public int smoothIterations = 5;

    [Header("Agent")]
    public GameObject agentPrefab;

    [Header("Camera")]
    public CameraController cameraController;

    private int[,] map;

    void Start()
```

```
{
    GenerateMap();
    cameraController.AdjustCamera(width, height);
}

void GenerateMap()
{
    if (useRandomSeed)
        seed = Random.Range(0, 100000);

    map = new int[width, height];

    GenerateBaseTerrain();
    AddBorder();
    ProcessMap();
    ValidateMap();
    RenderMap();
    SpawnAgent();
}

void GenerateBaseTerrain()
{
    float[,] noiseMap = NoiseGenerator.GenerateNoiseMap(
        width, height, seed, 20f
    );

    for (int x = 0; x < width; x++)
    {
```

```
        for (int y = 0; y < height; y++)
        {
            map[x, y] = noiseMap[x, y] > 0.5f ? 1 : 0;
        }
    }
}
void ProcessMap()
{
    map = CellularAutomata.SmoothMap(
        map,
        width,
        height,
        smoothIterations
    );
}
void AddBorder()
{
    for (int x = 0; x < width; x++)
    {
        map[x, 0] = 1;
        map[x, height - 1] = 1;
    }

    for (int y = 0; y < height; y++)
    {
        map[0, y] = 1;
        map[width - 1, y] = 1;
    }
```

```
}

void ValidateMap()
{
    map = ConnectivityChecker.EnsureConnectivity(
        map,
        width,
        height
    );
}
void RenderMap()
{
    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            GameObject tile = GameObject.CreatePrimitive(PrimitiveType.Cube);
            tile.transform.position = new Vector3(x, 0, y);

            if (map[x, y] == 0)
                tile.GetComponent<Renderer>().material.color = Color.white;
            else
                tile.GetComponent<Renderer>().material.color = Color.black;
        }
    }
}
void SpawnAgent()
{
```

```
    Vector2Int start = GetRandomWalkableTile();
    Vector2Int target = GetRandomWalkableTile();

    GameObject agent = Instantiate(
        agentPrefab,
        new Vector3(start.x, 1f, start.y),
        Quaternion.identity
    );

    var controller = agent.GetComponent<AgentController>();

    List<Vector2Int> path = AStarPathfinder.FindPath(
        map, width, height, start, target
    );

    controller.SetPath(path);
}
Vector2Int GetRandomWalkableTile()
{
    List<Vector2Int> walkableTiles = new List<Vector2Int>();

    for (int x = 1; x < width - 1; x++)
    {
        for (int y = 1; y < height - 1; y++)
        {
            if (map[x, y] == 0)
                walkableTiles.Add(new Vector2Int(x, y));
        }
```

```
        }

        if (walkableTiles.Count == 0)
            return new Vector2Int(1, 1);

        return walkableTiles[Random.Range(0, walkableTiles.Count)];
    }
}
```

NoiseGenerator.cs(Генерація шуму Перліна)

```
using UnityEngine;

public static class NoiseGenerator
{
    public static float[,] GenerateNoiseMap(int width, int height, int seed, float scale)
    {
        float[,] noiseMap = new float[width, height];

        System.Random prng = new System.Random(seed);

        float offsetX = prng.Next(-100000, 100000);
        float offsetY = prng.Next(-100000, 100000);

        for (int x = 0; x < width; x++)
        {
            for (int y = 0; y < height; y++)
            {
                float sampleX = (x + offsetX) / scale;
                float sampleY = (y + offsetY) / scale;
```

```
                float noiseValue = Mathf.PerlinNoise(sampleX, sampleY);
                noiseMap[x, y] = noiseValue;
            }
        }
        return noiseMap;
    }
}
```

CellularAutomata.cs(Клітинна автомата)

```
using UnityEngine;

public static class CellularAutomata
{
    public static int[,] SmoothMap(int[,] map, int width, int height, int iterations)
    {
        for (int i = 0; i < iterations; i++)
        {
            map = DoSimulationStep(map, width, height);
        }
        return map;
    }

    static int[,] DoSimulationStep(int[,] map, int width, int height)
    {
        int[,] newMap = new int[width, height];

        for (int x = 0; x < width; x++)
        {
```

```
            for (int y = 0; y < height; y++)
            {
                int wallCount = GetSurroundingWallCount(map, x, y, width, height);

                if (wallCount > 4)
                    newMap[x, y] = 1;
                else if (wallCount < 4)
                    newMap[x, y] = 0;
                else
                    newMap[x, y] = map[x, y];
            }
        }
        return newMap;
    }

    static int GetSurroundingWallCount(int[,] map, int gridX, int gridY, int width, int
height)
    {
        int wallCount = 0;

        for (int x = gridX - 1; x <= gridX + 1; x++)
        {
            for (int y = gridY - 1; y <= gridY + 1; y++)
            {
                if (x >= 0 && x < width && y >= 0 && y < height)
                {
                    if (x != gridX || y != gridY)
                        wallCount += map[x, y];
                }
```

```
            else
            {
                wallCount++;
            }
        }
    }
    return wallCount;
  }
}
```