МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
ФАКУЛЬТЕТ МЕХАТРОНІКИ ТА КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА
на тему:

Розроблення програмних методів процедурної генерації на прикладі roguelike

Рівень вищої освіти	другий (магістерський)
Спеціальність 122	Комп'ютерні науки
Освітня програма	Комп'ютерні науки

Виконав: студент групи МгІТ1-23
Москаленко І.А.
Науковий керівник:
к.т.н., доц. Астістова
Рецензент:
к.т.н., доц. Колиска О.З.

Київ 2024

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет Мехатроніки та комп'ютерних технологій
Кафедра Комп'ютерних наук
Рівень вищої освіти другий (магістерський)
Спеціальність 122 Комп'ютерні науки
Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри КН

_____ Наталія ЧУПРИНКА

«_____» _____ 2024 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Москаленко Ігорю Андрійовичу

1. Тема кваліфікаційної роботи: Розроблення програмних методів процедурної генерації на прикладі roguelike

Науковий керівник роботи к.т.н., доц., Астістова Тетяна Іванівна

затверджені наказом КНУТД від "03" 09 2024 року № 188-уч.

2. Вихідні дані до кваліфікаційної роботи: Розробки кафедри комп'ютерних наук, рекомендована література.

3. Зміст кваліфікаційної роботи: Розділ1. Аналіз предметної області; Розділ 2. Основні методи процедурної генерації; Розділ 3. Розробка програмного продукту.

4. Дата видачі завдання 30.09.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	20.08.2024	
2	Розділ 1 Аналіз предметної області	01.09.2024	
3	Розділ 2. Основні методи процедурної генерації	10.10.2024	
4	Розділ 3. Розробка програмного продукту.	25.10.2024	
5	Висновки	29.10.2024	
6	Оформлення кваліфікаційної роботи (чистовий варіант)	10.10.2024	
7	Подача кваліфікаційної роботи науковому керівнику для відгуку	12.11.2024	
8	Подача кваліфікаційної роботи для рецензування	18.11.2024	
9	Перевірка кваліфікаційної роботи на наявність ознак плагіату	20.11.2024	
10	Подання кваліфікаційної роботи на затвердження завідувачу кафедри	22.11.2024	

Студент _____ Ігор МОСКАЛЕНКО

Науковий керівник _____ Тетяна АСТІСТОВА

АНОТАЦІЯ

Москаленко І. А. Розроблення програмних методів процедурної генерації на прикладі roguelike.

Кваліфікаційна робота за спеціальністю 122 — «Комп’ютерні науки» — Київський національний університет технологій та дизайну, Київ, 2024 рік.

У кваліфікаційній роботі було розроблено програму що використовує методи процедурної генерації для автоматизованого процесу створення унікального контенту. Програма використовує алгоритми для псевдовипадкової генерації для спрощення процесу розробки та підвищення унікальності генерованого контенту в процесі експлуатації.

Реалізовано виконання програми, заснованої на принципах жанру roguelike, в основі якого лежить псевдовипадкова генеративність. Розробка базується на кількох основних методах, які застосовуються у різних аспектах генерації, але об’єднані в одній програмі для покращення генеративних особливостей.

Ключові слова: процедурна генерація, roguelike, ігрові рівні, генерація рівнів, dungeon generation, випадкова генерація, геймдев, Unity, Bifrost, C#

ABSTRACT

Moskalenko I. A. Development of software methods of procedural generation on the example of roguelike.

Qualification work in the specialty 122 – “Computer Science” – Kyiv National University of Technology and Design, Kyiv, 2024.

In qualification work, a program utilizing procedural generation methods for automating the creation of unique content was developed. The program employs algorithms for pseudorandom generation to streamline the development process and enhance the uniqueness of generated content during operation.

The program's implementation is based on the principles of the roguelike genre, which relies on pseudorandom generation as its foundation. The development integrates several core methods in various generation aspects, unified within a single program to improve generative capabilities.

Keywords: procedural generation, roguelike, game levels, level generation, dungeon generation, random generation, gamedev, Unity, Bifrost.

ЗМІСТ

ВСТУП

Актуальність роботи: сьогодні існує попит на підвищення автоматизації та спрощення різноманітних алгоритмів виконання тих чи інших задач. Водночас збільшується попит на велику кількість контенту, високу швидкість його створення і при цьому збереження унікальності. Для вирішення цих питань ідеально підходять методи процедурної генерації контенту, які допомагають швидко і у великих об’ємах згенерувати більшість видів контенту. Контент генерується майже випадково, з можливістю задання певних правил, для корекції згенерованих даних. Такі алгоритми вже широко використовуються і набувають все більшої популярності.

Мета досліджень: метою є дослідити предметну область у сфері процедурної генерації, яким чином і завдяки яким чинникам такі алгоритми набули популярності. Проаналізувати методи процедурної генерації в розробці розважального контенту, як в найбільш поширеній сфері їх застосування в процесі розробки. Дослідити найбільш поширені алгоритми та напрями їх застосування. Розробити програмні методи виконання процедурної генерації на прикладі програмного забезпечення з жанру roguelike.

Об’єкт дослідження: процедурна генерація її розвиток та застосування в різних сферах розробки контенту.

Предмет дослідження: розроблення програмного забезпечення з методами процедурної генерації орієнтованого на жанр roguelike.

Елементи наукової новизни: комбінування кількох різних методів процедурної генерації в одному проєкті у різних напрямах розробки.

Практична значущість роботи: створена програма, в популярному жанрі, що використовує передові методи генерації контенту, а також комбінує різні напрями застосування цих методів у процесі розробки.

Апробація результатів роботи: програма має високий потенціал різноманітності контенту завдяки алгоритмам процедурної генерації, що дозволяють створювати унікальний контент у різних аспектах розробки.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальне уявлення процедурної генерації

Процедурна генерація – це метод створення даних алгоритмічно, а не вручну, зазвичай шляхом поєднання створеного людиною контенту та алгоритмів, у поєднанні з комп’ютерною випадковістю та обчислювальною потужністю. У комп’ютерній графіці її часто використовують для створення текстур і 3D-моделей. У відеоіграх, для автоматичного створення великої кількості контенту в грі. Залежно від реалізації, переваги процедурної генерації можуть включати менші розміри файлів, більшу кількість контенту та випадковість для більш різноманітного геймплею.

Термін "процедурний" відноситься до процесу, який обчислює певну функцію. Звичайним процедурним контентом є текстури та меші, на Рис. 1.1 зображено типові процедурно згенеровані текстури. Звук також часто генерується процедурно і має застосування як у синтезі мови, так і в музиці. Він використовувався для створення композицій у різних жанрах електронної музики такими артистами, як Браян Іно, який популяризував термін "генеративна музика".

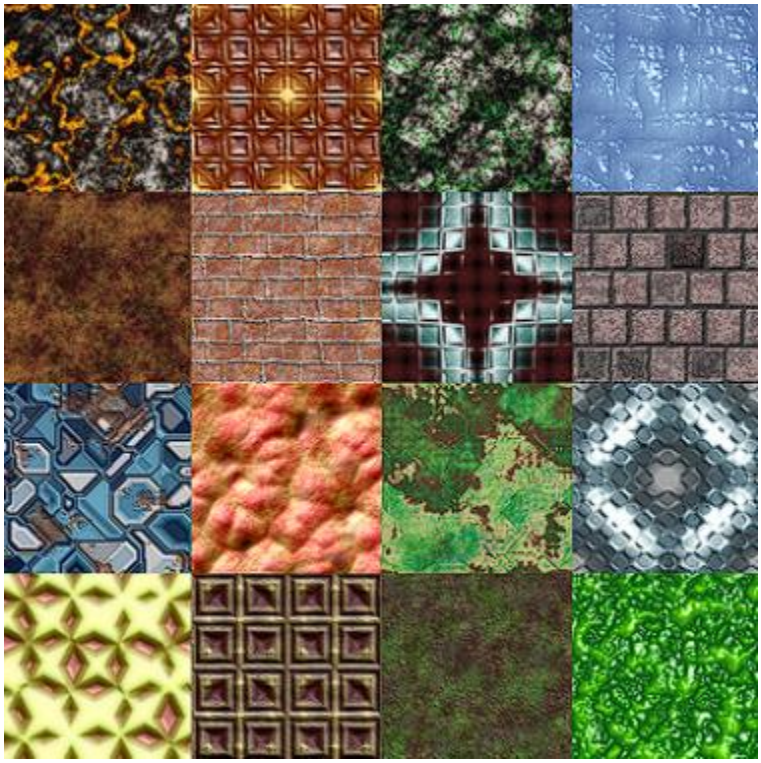


Рис. 1.1 – Процедурно згенеровані текстури

Хоча розробники програмного забезпечення застосовують методи

процедурної генерації протягом багатьох років, лише деякі продукти широко використовують цей підхід. Процедурно згенеровані елементи з’являлися в ранніх відеоіграх: The Elder Scrolls II: Daggerfall (1996 рік) відбувається у переважно процедурно згенерованому світі, приблизно дві третини розміру Британських островів. Soldier of Fortune (2000 рік) від Raven Software використовує прості процедури для деталізації моделей ворогів, а його продовження містило режим випадково згенерованих рівнів. Avalanche Studios використовувала процедурну генерацію для створення великої та різноманітної групи деталізованих тропічних островів для Just Cause (2006 рік). Гра No Man’s Sky (2016 рік), розроблена ігровою студією Hello Games, повністю базується на процедурно згенерованих елементах.

Сучасна міжнародна субкультура цифрового мистецтва demoscene (демо-сцена), яка зосереджена на створенні демонстрацій, або так званих демоверсій, використовує процедурну генерацію для пакування великої кількості аудіовізуального контенту у відносно невеликі програми. Мета таких демоверсій – продемонструвати навички програмування, образотворчого мистецтва та музики, не розробляючи для цього повноцінний великий проєкт.

Нові методи та застосування процедурної генерації щорічно представляються на таких конференціях, як IEEE (Institute of Electrical and Electronics Engineers) Conference on Computational Intelligence and Games та AAAI (The Association for the Advancement of Artificial Intelligence) Conference on Artificial Intelligence and Interactive Digital Entertainment.

У застосуванні процедурної генерації у відеоіграх, які призначені для так званої реграбельності (високого показника повторних проходжень), існують побоювання, що процедурні системи можуть генерувати нескінченну кількість світів для дослідження, але без достатнього керівництва людини та правил для їхнього впорядкування, в результаті отримаємо "процедурну вівсянку". Цей термін, придуманий письменницею Кейт Комптон, має на увазі, що хоча математично можливо згенерувати тисячі мисок вівсянки за допомогою процедурної генерації, користувач сприйматиме їх як однакові і їм бракуватиме поняття відчутної унікальності, до якої повинна прагнути процедурна система.

Відмінність процедурної генерації від випадкової полягає в тому, що процес процедурної генерації можна контролювати правилами та алгоритмами, таким чином впливаючи на результат та відсоток детермінованості чи умовної випадковості даних. Саме ця можливість контролю та обмеження випадковості, робить процедурну генерацію зручним та необхідним інструментом для створення та обробки великих масивів даних.

1.2 Витоки застосування в іграх

Використання процедурної генерації в іграх виникло ще в настільних рольових іграх, які власне і започаткували відомий жанр комп'ютерних ігор RPG (role-play game), серед них, зокрема, і легендарна «Dungeons & Dragons» (скорочено «D&D» чи «DnD»).

«Dungeons & Dragons» або «Підземелля і дракони» це найперша та найпопулярніша настільна гра у світі. Вперше опублікована у 1974 році, вона перевидається новими редакціями та залишається популярною донині, ставши одним з канонів поп-культури. На Рис 1.2 зображено типовий настільний набір для гри.

Рис 1.2 – Процес гри в «Dungeons & Dragons»

В «DnD» ігровий процес заснований на кидках гральних кубиків, які визначають ступінь випадковості, а «Dungeon Master» («Володар підземелля» або «Майстер гри») гравець, який керує грою, і фактично є суперником всіх

інших гравців, виступає своєрідним комп’ютером, який обчислює: як випадкове число з кубиків і сталі характеристики персонажів гравців впливають на будь-яку ігрову подію. Наприклад, порівняння характеристик гравця і суперника у битві, перевірка здібності для певної дії – все визначається сталим числом, додатковим випадковим числом і операцією над ними, згідно правил гри або рішення «Володаря підземелля». Зокрема, під час гри фігурки гравців пересуваються картою підземелля, яку «Dungeon Master» малює власноруч, він вирішує де і що на ній розміщується, який вигляд має саме підземелля, що там очікує гравців та майже всі інші аспекти гри.

В подальших редакціях гри з’явились нові правила, зокрема, провідна настільна система Advanced Dungeons & Dragons, яка надала способи для «Майстра гри» створювати підземелля та місцевість за допомогою випадкових кидків кубиків, розширених у пізніших виданнях складними розгалуженими процедурними таблицями. Це дозволило створювати більш уніфіковану та логічну структуру підземелля, де кожен рівень та кожен елемент могли бути генеровані випадково, але згідно з певними правилами.

Згодом було випущено комп’ютерну програму Dungeon Master's Assistant, яка створювала підземелля на основі цих опублікованих таблиць. Інші настільні рольові ігри запозичили подібні концепції процедурної генерації для різних елементів світу. Зараз багато онлайн-інструментів для Dungeon Masters різною мірою використовують процедурну генерацію.

1.3 Roguelike

До появи графічно орієнтованих відеоігор, ігри жанру roguelike, що були безпосередньо натхненні «Dungeons & Dragons» і адаптовані для одиночного проходження, активно використовували і просували процедурну генерацію для випадкового створення підземель, аналогічно до того, як це робили настільні ігри. До таких ранніх ігор належать Beneath Apple Manor (1978 рік) та безпосередньо засновник жанру Rogue (1980 рік).

Rogue не була першою грою натхненною «Dungeon&Dragons» але її визначною особливістю стало використання схожого з «DnD» принципу випадкової побудови ігрових мап, але замість кубиків застосовувались

алгоритми процедурної генерації.

У Rogue гравці керують персонажем, досліджуючи кілька рівнів підземелля, у пошуках так званого Амулету Єндора, який розташований на найнижчому рівні підземелля. Гра починається на самому верхньому рівні підземелля з безліччю монстрів і скарбів. Мета полягає в тому, щоб пройти шлях до нижнього рівня, отримати амулет, а потім піднятися на поверхню. Перемагати монстрів на рівнях стає все важче. Поки амулет не буде отримано, гравець не може повернутися на попередні рівні.

Кожен рівень має кілька кімнат, що з'єднані коридорами, деякі коридори можуть бути тупиковими для урізноманітнення геймплею. В кімнатах можуть знаходитись гральні предмети та монстри, а також в одній з кімнат розташовується перехід на наступний рівень, який, власне, гравцеві і потрібно знайти. Всі ці елементи, їх кількість, розміщення та послідовність визначаються псевдо випадково за допомогою процедурної генерації. На Рис. 1.3 зображено типову будову рівня у Rogue.

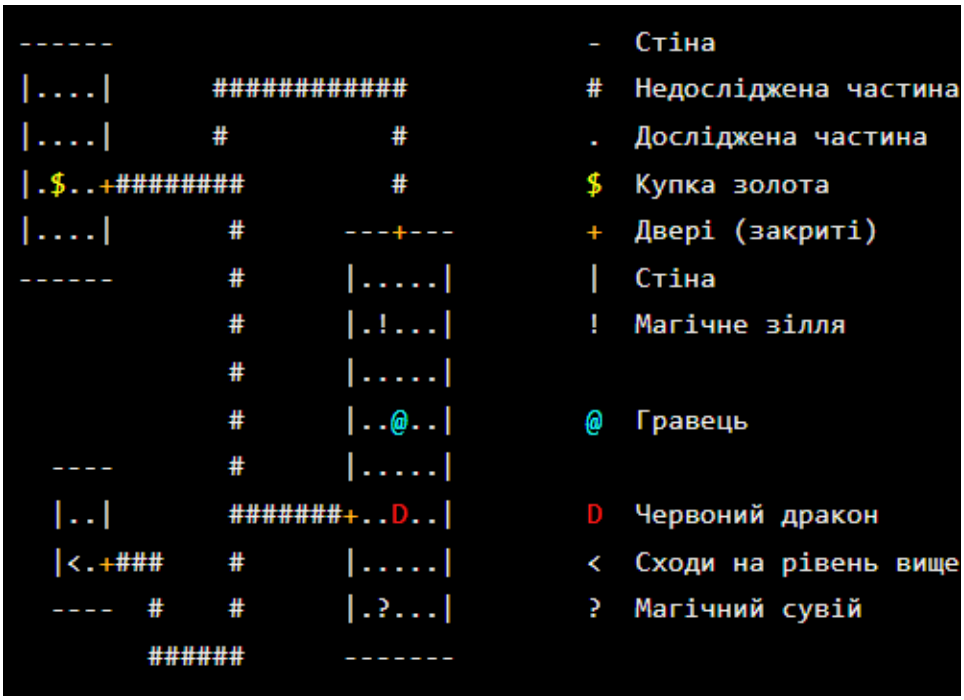


Рис 1.3 – схема побудови підземелля у Rogue з ASCII-графікою

На початку кожного рівня, гравець не знає де знаходяться інші кімнати та коридори, йому треба випадково знайти їх пересуваючись персонажем у просторі, після чого вони промалюються і доповнять загальну картину. Rogue покрокова гра, події відбуваються на квадратній сітці, представленій у ASCII,

що дає гравцям час визначити найкращий хід, щоб вижити. Персонаж гравця повинен відбиватися від безлічі монстрів, які блукають підземеллями. По дорозі гравці можуть збирати скарби, які можуть допомогти їм у нападі чи обороні, наприклад, зброю, обладунки, зілля, сувої та інші магічні предмети.

Rogue реалізує permadeath (перманентна, остаточна смерть) як вибір дизайну, щоб зробити кожну дію гравця значущою. Якщо персонаж гравця втратить усе своє здоров'я в бою чи іншим способом, гру буде закінчено. Після чого гравець повинен перезапустити гру з новим персонажем, оскільки мертвий персонаж не може відродитися або бути повернутий перезавантаженням зі збереження. Крім того, жодна гра не схожа на будь-яку попередню, оскільки рівні підземелля, зустрічі з монстрами та скарби генеруються процедурно для кожного проходження.

Ці дві особливості, перманентна смерть та процедурна генерація рівнів, стали основою для цілого під-жанру рольових відеоігор, які також взяли похідну назву від Rogue, і звуться roguelike або Rogue-подібними іграми.

Система процедурної генерації в roguelike іграх створювала підземелля з допомогою текстових ASCII-символів (або, вже пізніше, в піксельних графічних системах), визначаючи кімнати, коридори, монстрів та скарби, щоб створити виклики для гравця. roguelike ігри та ігри, що базуються на їх концепціях, дозволяють розвивати складний ігровий процес, не витрачаючи занадто багато часу на створення світу гри.

1.4 Берлінське тлумачення

Те, які елементи геймплею чітко визначають гру жанру roguelike, залишаються предметом обговорення у ігровій спільноті. Існує загальне визнання, що roguelike ігри включають елементи геймплею, популяризовані текстовою грою Rogue (1980), яка зазнала багатьох варіацій через свій успіх. Нині, в каталозі Steam доступно кілька сотень ігор, що визначають себе як roguelike, а користувацька вікі RogueBasin відстежує сотні таких ігор і їхній розвиток.

Деякі гравці та розробники намагалися визначити більш вузьке тлумачення терміну roguelike, оскільки варіації на тему Rogue вводили нові

концепції або відмовлялися від інших принципів, які, на їхню думку, відходили від того, що робило Rogue саме Rogue. На Міжнародній конференції розробників roguelike ігор 2008 року в Берліні учасники встановили визначення для roguelike ігор, відоме як "Берлінське тлумачення". Це тлумачення включало набір високовартісних і низьковартісних факторів, ґрунтуючись на п'яти канонічних іграх roguelike: ADOM, Angband, Linley’s Dungeon Crawl, NetHack та Rogue. Берлінське тлумачення було розроблене, щоб визначити, "наскільки roguelike є та чи інша гра", зазначаючи, що відсутність одного з факторів не виключає гру з категорії roguelike, а наявність цих факторів не гарантує, що гра справді є roguelike.

Джон Харрис з Game Set Watch продемонстрував це, використовуючи ці критерії для числової оцінки кількох ігор, які могли бути порівняні з roguelike. Linley’s Dungeon Crawl та NetHack отримали найвищі бали, набравши 57,5 балів із 60 можливих за Берлінським тлумаченням, тоді як Toe Jam & Earl і Diablo, які часто порівнюють із roguelike, набрали лише близько половини балів.

Берлінське тлумачення визначило дев'ять факторів які несуть велике значення:

Випадкові підземелля

Генерація випадкових підземель для збільшення реграбельності гри: Ігри можуть включати попередньо визначені рівні, такі як місто, яке є характерним для родини Moria, де гравець може купувати і продавати обладнання, але такі рівні зменшують елемент випадковості, який визначається Берлінським тлумаченням. «Випадкова генерація» зазвичай базується на процедурному підході, а не на справжній випадковості. Процедурна генерація використовує набір правил, визначених розробниками гри, для ініціалізації генерації підземелля, щоб забезпечити можливість проходження кожного рівня без спеціального обладнання і створити естетично приємні рівні. Крім того, зовнішній вигляд магічних предметів може змінюватися від одного проходження до іншого — наприклад, "пухирчасте" зілля може спочатку лікувати рани, а наступного разу отруїти персонажа.

Перманентна смерть

Після смерті персонажа гравець має почати нову гру, що називається «проходженням» або «забігом», і рівні гри будуть генеровані заново завдяки процедурній генерації. Функція «збереження гри» лише призупиняє ігровий процес, але не дозволяє відновити стан без обмежень; збережені сесії видаляються після відновлення або смерті персонажа. Гравці можуть обійти це, створивши резервну копію збережених ігрових даних, що зазвичай вважається шахрайством; розробники Rogue представили функцію permadeath після введення функції збереження, виявивши, що гравці неодноразово завантажували збережені ігри для досягнення найкращих результатів, як зазначив Майкл Той, один із розробників Rogue, вони не бачили мету в тому, щоб зробити гру болючою чи важкою, а скоріше хотіли, щоб кожне рішення гравця мало вагу, створюючи більш поглиблений досвід.

Розглянемо термінологію, яку ми будемо використовувати в нашій роботі та яка прийнята в цій сфері та деякі правила , які притаманні в іграх.

- Покроковий геймплей.

Гра є покроковою, що дає гравцеві стільки часу, скільки йому потрібно для прийняття рішення. Геймплей зазвичай базується на кроках, де дії гравця виконуються по черзі, і для завершення кожної дії потрібно змінний час в грі. Процеси в грі (наприклад, рух монстрів і взаємодії, прогресивні ефекти, такі як отруєння або голод) розвиваються на основі часу, що проходить під впливом цих дій.

- Ігрова сітка

Гра проходить на рівномірній сітці плиток, яка зазвичай представлена в ASCII-форматі для підземелля.

- Немодальність гри

Гра є немодальною, тобто кожна дія повинна бути доступною гравцеві незалежно від того, де він знаходиться в грі. Однак, за Берлінським тлумаченням, магазини, як у Angband, порушують цю немодальність.

- Складність гри

Гра має певний ступінь складності через кількість різних ігрових систем, які дозволяють гравцеві досягати певних цілей кількома способами, створюючи неперевершений геймплей. Наприклад, щоб пройти через замкнені двері, гравець може спробувати зламати замок, вибити його ногою, спалити двері або навіть прокласти тунель навколо них, залежно від поточної ситуації та інвентарю. Загальновідома фраза, пов'язана з NetHack, звучить як «Команда розробників думає про все», що відображає те, як розробники, здається, передбачили кожну можливу комбінацію дій, яку гравець може спробувати у своїй стратегії гри. Наприклад, можна використовувати рукавички для захисту персонажа, коли він тримає труп Василіска як зброю, яка перетворює ворогів на камінь одним дотиком.

- Управління ресурсами для виживання

Гравець повинен використовувати управління ресурсами, щоб вижити. Предмети, які допомагають підтримувати здоров’я гравця, такі як їжа та цілющі предмети, обмежені, і гравець повинен з’ясувати, як використовувати їх найбільш вигідно, щоб вижити в підземеллі. Крім того, USGamer розглядає «зниження витривалості» як ще одну функцію, пов’язану з управлінням ресурсами. Персонажу гравця постійно потрібно шукати їжу, щоб уникнути голоду, що заважає гравцеві використовувати відновлення здоров'я, просто проходячи ходи протягом тривалого періоду часу або борючись із дуже слабкими монстрами в підземеллях низького рівня. Річ Карлсон, один із творців раннього roguelike – Strange Adventures in Infinite Space, назвав цей аспект свого роду «годинником», накладаючи певний термін або обмеження на те, скільки гравець може досліджувати, і створюючи напругу в грі.

- Геймплей в стилі «Hack & Slash»

Гра зосереджена на геймплеї, заснованому на так званій «рубанині» ворогів, де метою є вбити багато монстрів, і де інших мирних варіантів не існує.

- Дослідження світу та виявлення предметів

Гравець повинен досліджувати світ гри та виявляти призначення незнайомих предметів. У іграх з випадковою генерацією це потрібно робити знову при кожному новому проходженні, оскільки і карта, і зовнішній вигляд

предметів змінюються.

Фактори низького значення за Берлінським тлумаченням включають:

1. Гра базується на контролі лише одного персонажа протягом одного проходження.

2. Монстри мають поведінку, схожу на поведінку персонажа-гравця, наприклад, здатність підбирати предмети та використовувати їх або надавати заклинання.

3. Гра спрямована на надання тактичного виклику, що може вимагати від гравців кількох проходжень, щоб навчитися відповідним тактикам для виживання.

4. Гра передбачає дослідження підземель, які складаються з кімнат і з'єднаних коридорів. Деякі ігри можуть мати відкриті простори або природні особливості, такі як річки, але це вважається порушенням Берлінського тлумачення.

5. Гра відображає статус гравця та процес гри через числа на екрані або інтерфейсі гри.

Хоча це не обговорюється безпосередньо в Берлінському тлумаченням, roguelike ігри зазвичай є однокористувацькими. У багатокористувацьких системах таблиці лідерів часто діляться між гравцями. Деякі roguelike ігри дозволяють залишки колишніх персонажів гравців з'являтися в пізніших сесіях гри у вигляді привидів або могильних знаків. Наприклад, в NetHack колишні персонажі можуть навіть з'являтися як вороги в підземеллі. Існують також багатокористувацькі покрокові похідні, такі як TomeNET, MAngband і Crossfire, які можна грати онлайн.

У своїй статті «Скинь Берлінське тлумачення!» 2013 року, розробник Даррен Грей звинувачує «Берлінське тлумачення» у неточності, застарілості та неприязні до яскравого й відкритого жанру. Зокрема, він висміює такі елементи, як ASCII та підземелля, вважаючи їх неактуальними для жанру, який традиційно ставить акцент на механіку гри, а не на естетику чи сюжет. Він критикує загальне використання Берлінського тлумачення як єдиного визначення roguelike ігор.

У більш широкій ігровій спільноті термін roguelike або «з елементами roguelike» частіше використовується для опису будь-якої гри, яка поєднує перманентну смерть з процедурно згенерованим контентом.

1.5 Порівняльна характеристика Roguelike та Rogue-lite

З розвитком комп'ютерної техніки та відеоігор, здатних на більш складну графіку та геймплей, з'явилося багато ігор, які вільно базуються на класичному дизайні roguelike, але відрізняються однією або кількома особливостями. Багато з цих ігор використовують концепції процедурно згенерованих мап та перманентної смерті, але відходять від покрокового геймплею та руху по плитковій сітці, часто інтегруючи інші жанри, такі як екшн-ігри чи платформери.

Інші ігри, що походять від roguelike, засновані на спостереженні, що традиційні roguelike ігри є надзвичайно складними з крутою кривою навчання. На Рис. 1.4 зображена діаграма росту навичок гравців у roguelike, необхідних для проходження гри.

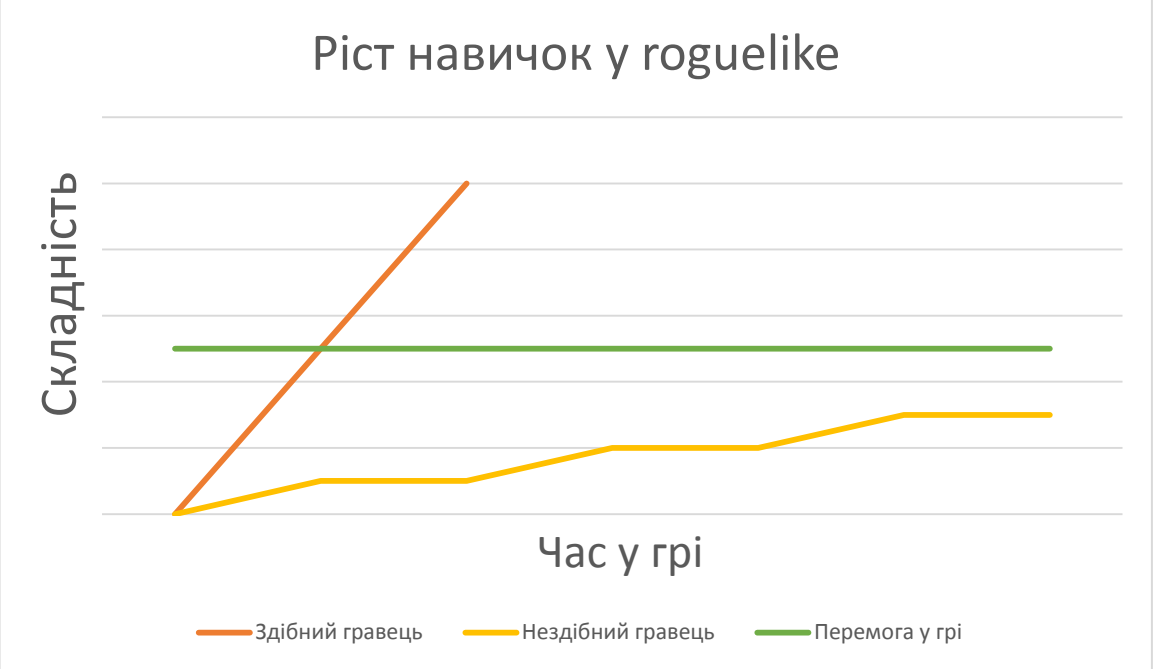


Рис. 1.4 – Діаграма росту навичок гравців у roguelike з часом

Складність залишається сталою, а рівень навичок гравців росте пропорційно проведеному у грі часі але також сильно залежить від здібностей конкретного гравця. Якщо гравець досить здібний, то він швидше навчиться, і не зважаючи на перезапуск прогресу після поразки, зможе так чи інакше досить

швидко подолати всі перешкоди і пройти гру до кінця. В той час, як не дуже здібний гравець може ніколи не отримати необхідних навичок або просто не встигнути їх розвинути за кілька сеансів, аж до моменту втрати інтересу до гри.

Така система складності робить традиційні roguelike ігри важкими для продажу ширшій аудиторії. Нові ігри часто включають елементи, які знижують складність, щоб залучити більшу кількість гравців.

Багато ігор, які включають деякі елементи Берлінського тлумачення, називають себе roguelike, але мають мало спільного з оригінальним Rogue, що спричиняє плутанину та розмивання терміну. Деякі гравці roguelike ігор, що відповідають Берлінському тлумаченню, не схвалюють розмивання цього терміну, вважаючи, що в 1990-х і 2000-х роках слово "roguelikes" добре відокремлювало ігри, які відмовлялися від естетики на користь глибини геймплею, від ігор, які більше нагадували інтерактивні фільми, зокрема ігор, що включали елементи реального часу, які знижували складність гри.

Тому терміни "rogue-lite" або "roguelike-like" (або roguelike-подібні) почали використовуватися деким для відокремлення ігор, що мають деякі, але не всі риси Берлінського тлумачення, від тих, що повністю відповідають визначенню roguelike за Берлінським тлумаченням. Фраза "процедурний лабіринт смерті" також застосовується до таких ігор, оскільки вони зберігають ідею перманентної смерті та випадкової генерації рівнів, але не мають інших високовартісних факторів, які зазвичай асоціюються з іграми roguelike.

Rogue-lite ігри віддають перевагу коротким сеансам гри з умовами перемоги, на відміну від деяких традиційних roguelike ігор, які можуть гратися безкінечно. Коротка тривалість одного сеансу гри в rogue-lite іграх може мотивувати гравців постійно перегравати гру в надії на досягнення завершення, що робить повторюваність важливим фактором у цих іграх.

Ігровий журналіст Джошуа Байсер зазначив, що кілька ігор, які вважаються rogue-lite, містять фіксовані події, навіть якщо спосіб досягнення цих подій відбувається через процедурну генерацію. У той час як roguelike гра зазвичай не має такої передбачуваності. Наприклад, деякі rogue-lite ігри вимагають від гравця подорожувати фіксованою кількістю біомів, кожен з яких

завершується бійкою з босом, як у Rogue Legacy (2012 рік).

Пов'язані з їхньою короткою тривалістю, багато rogue-lite ігор мають так звану «метагру» (гру у грі), де досягнення певних цілей відкриває постійні можливості, такі як вибір нового персонажа на початку гри або додавання нових предметів і монстрів у процедурну генерацію рівнів. Альтернативно, кожен прохід через rogue-lite може бути спрямований на збирання ресурсів, які потім використовуються для розвитку персонажа в межах метагри. Гравець може достроково відмовитися від завершення «забігу», коли зібрано достатньо матеріалів для цього розвитку.

Такий принцип збереження прогресу та поступового покращення персонажу, що полегшує проходження гри, впливає на криву навчання досить цікавим чином (див Рис. 1.5). Замість сталого значення необхідного рівня навичок для проходження гри, цей параметр зменшується з часом, роблячи гру більш складною на перших етапах, і навпаки легшою на пізніх, що у купі з елементами метагри зберігає цікавість та дозволяє менш здібним гравцям з більшою ймовірністю проходити гру до кінця.

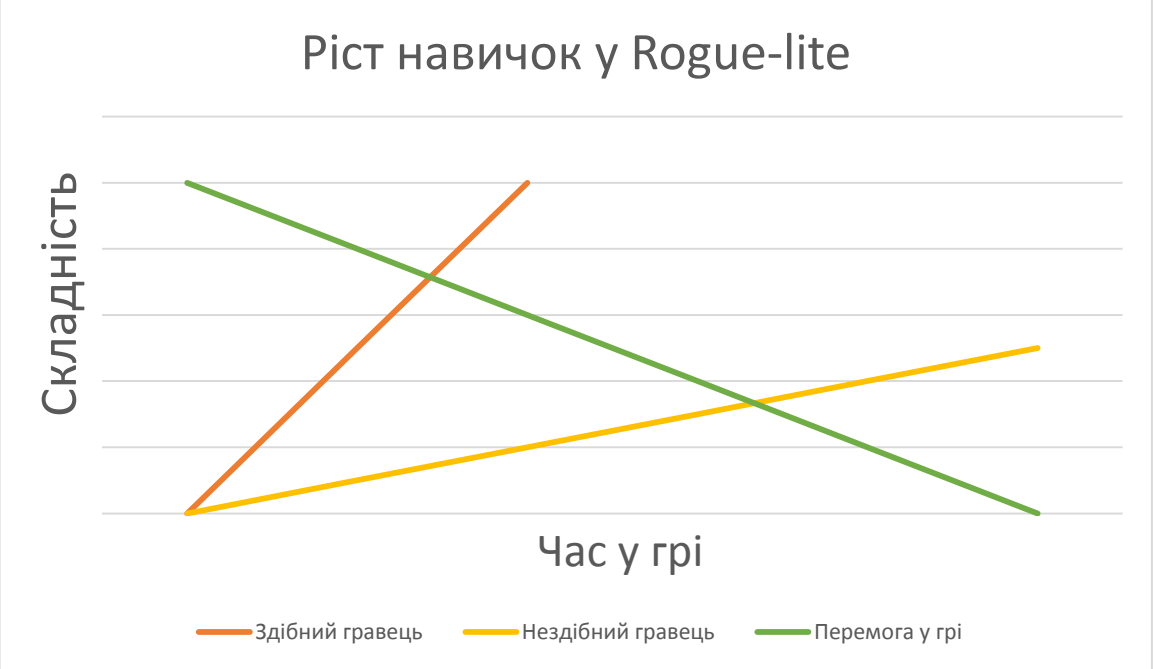


Рис. 1.5 – Діаграма росту навичок гравців у Rogue-lite з часом

Багато rogue-lite ігор пропонують щоденні завдання, у яких використовується попередньо задане так зване «випадкове насіння» (random seed) для генерації рівнів гри в детермінований спосіб, так що кожен гравець

матиме однакові зустрічі та виклики. Гравці намагаються пройти гру через ці рівні або отримати найвищий бал для онлайн таблиці лідерів.

Rogue-lite ігри також можуть дозволяти гравцям вводити випадкове насіння безпосередньо, щоб мати змогу повторно пройти той самий набір рівнів або поділитися складним набором рівнів з іншими гравцями.

З огляду на популярність roguelike ігор, які відхиляються від Берлінського тлумачення, зокрема Rogue-lite, з'явилися кілька під жанрів.

Action-roguelike зазвичай поєднує механіку екшн-ігор з roguelike елементами, замінюючи покроковий геймплей на реальний час. Spelunky (2008 рік) є прикладом поєднання платформерменості з формулою roguelike, тоді як The Binding of Isaac (2011 рік) та Enter the Gungeon (2016 рік) — це успішні екшн-ігри з елементами шутера в стилі roguelike. В межах action-roguelike також з'явився shooter-roguelike, де Vampire Survivors (2022 рік) є провідним прикладом: у таких іграх гравець зазвичай бореться з хвилями ворогів, при цьому персонаж автоматично використовує всі можливі атаки без втручання гравця, і можливість покращувати свого персонажа через випадковий вибір покращень після кожної перемоги над ворогами.

Інший вид жанру такого виду є roguelike — це deck-builder (побудова колоди), де бій вирішується за допомогою карт або еквівалентних об'єктів. Ці ігри натхненні фізичними живими картковими іграми, де гравець будує свою колоду протягом гри, що змушує планувати стратегію на ходу. Хоча гра Dream Quest 2014 року вважається першою такою відеогрою, популярність жанру була закріплена Slay the Spire у 2017 році.

1.6 Становлення Rogue-lite

Жанр roguelike пережив ренесанс на західних ринках після 2000 року завдяки так званим інді - розробникам (незалежним невеликим студіям чи розробникам, проєкти яких так само називають «indie-games» або «інді - ігри»), які створили новий піджанр, що отримав назву «rogue-lite», хоча такі ігри іноді також називають «roguelike-подібними». Інді - розробники почали включати елементи roguelike в жанри, які зазвичай не асоціюються з цим жанром, створюючи ігри, які стали основою для цього нового піджанру.

Два з найперших прикладів rogue-lite ігор — це Strange Adventures in Infinite Space (2002) та його сиквел Weird Worlds: Return to Infinite Space (2005) від Digital Eel, обидві ігри про космічні дослідження, що включали випадково згенеровані планети і зустрічі з ворогами, а також перманентну смерть. Digital Eel базували свої роботи на грі Starflight та roguelike іграх, таких як NetHack, але хотіли створити коротший досвід, який було б легше перегравати, подібно до настільних ігор, як Deathmaze і The Sorcerer's Cave, які мають спільні елементи з roguelike іграми.

Гра Spelunky 2008 року, випущена незабаром після формування Берлінського тлумачення, вважається значним внеском у розвиток незалежно розроблених rogue-lite ігор. Розроблена Дереком Ю, Spelunky мала на меті поєднати глибину геймплею, притаманну roguelike іграм, з простотою та доступністю платформ. В результаті вийшла платформена гра, що включала концепцію перманентної смерті, де гравець веде персонажа-дослідника через випадково згенеровані печери. Метою було створити "глибокий" геймплей, в якому можна було б грати знову і знову, причому випадково згенеровані ситуації змушують гравця розробляти нові, нестандартні стратегії на ходу.

Розробник Джейсон Рорер зазначив, що Spelunky "повністю змінила моє розуміння дизайну однокористувацьких відеоігор". Едмунд МакМіллен, розробник The Binding of Isaac (2011), та Кенні і Тедді Лі, співрозробники Rogue Legacy (2012 рік), віддали належне підходу Дерека Ю зі Spelunky, який показав, як можна звести суть традиційного roguelike і застосувати її до інших жанрів, що вони й зробили у своїх rogue-lite іграх. Джастін Ма і Метью Девіс, спів розробники FTL: Faster Than Light (2012 рік), також відзначили як вплив на їх гру Weird Worlds: Return to Infinite Space та Spelunky.

Усі ці ігри здобули критичне визнання, і їхній успіх призвів до сучасного відродження rogue-lite ігор після їх випуску. Таке нове успішне відродження rogue-lite ігор вважається частиною більшої тенденції серед гравців, які активно грають як в настільні, так і в комп'ютерні ігри, шукаючи "глибокий ігровий досвід", як описав розробник «100 Rogues» Кіт Бергун. Цей досвід, на його думку, може не завжди пропонуватися популярними іграми.

Девід Бамгуарт з Gaslamp Games зазначив, що в rogue-lite іграх з випадковою генерацією та перманентною смертю є елемент ризику, який допомагає гравцеві глибше зануритися в долю свого персонажа: "Смертельна небезпека, властива невідомим середовищам roguelike, надає цьому інвестуванню велику значущість".

Більш того, багато нових rogue-lite ігор прагнуть зменшити високу складність та безжалісність, які були притаманні традиційним roguelike іграм, надаючи новим гравцям більше допомоги через користувацькі гіди та проходження, які стали доступними завдяки широкому доступу до Інтернету.

Фаб'єн Фішер вважає, що гравці почали звертатися до незалежно розроблених rogue-lite ігор, оскільки вони втомилися від "поверхневого геймплею, надмірної уваги до видовищ та божевільного контенту у іграх, створених великими AAA-розробниками та видавцями.

МакМіллен, розробник The Binding of Isaac, зазначив, що впровадження елементів roguelike в інші механіки ігор може бути складним через складні інтерфейси, властиві roguelike іграм. Однак він підкреслив, що зрештою це призводить до "все більш красивого, глибокого та вічного дизайну, який дозволяє створити, здавалося б, динамічний досвід для гравців, так що кожного разу, коли вони грають у вашу гру, вони отримують абсолютно нову пригоду".

Процедурно згенерований світ дозволяє розробникам створювати величезну кількість контенту для гри, не витрачаючи ресурси на проектування детальних світів.

Приклади успішних ігор, які інтегрували елементи roguelike в інші жанри, включають:

- Dead Cells (2018 рік) – roguelike у поєднанні з платформером в стилі Metroidvania.
- Slay the Spire (2017 рік) – перенесення прогресії roguelike в гру з механікою побудови колоди карт.
- Crypt of the NecroDancer (2015 рік) – використання ритмічної гри в roguelike підземеллях.
- Enter the Gungeon (2016 рік) – встановлення прогресії roguelike в

шутері shoot 'em up.

- Vampire Survivors (2022 рік) – мінімалістичний roguelike шутер shoot 'em up.
- Hades (2020 рік) та Hades II (2024 рік) – roguelite екшн-рольові ігри, які сильно інтегрують нелінійний наратив, даючи гравцеві причину постійно повертатися до гри, і допомогли залучити гравців до жанру roguelike, яких раніше відштовхувала висока складність.

1.7 Сучасне застосування процедурної генерації

Хоча застосування процедурної генерації для створення унікальних рівнів у далекій Rogue 1980 року і стала переломним моментом, це не було єдиним напрямом застосування даної технології у геймдеві.

Деякі ігри використовували псевдовипадкові генератори чисел (PRNG). Ці PRNG часто застосовувалися з попередньо визначеними значеннями насіння для створення дуже великих ігрових світів, які здавалися попередньо створеними. Наприклад, The Sentinel (1986 рік) нібито містив 10,000 різних рівнів, зберігаючи їх всього на 48 та 64 кілобайтах пам'яті. В екстремальному випадку Elite (1984 рік) спочатку планувалася для включення 248 галактик (приблизно 282 трильйони) з 256 сонячними системами в кожній. Однак видавець побоювався, що настільки велика всесвіт може викликати невіру в гравців, і для фінальної версії було обрано лише 8 галактик, по 256 згенерованих зоряних систем (див Рис. 1.6).

Рис. 1.6 – Процедурно згенеровані зоряні системи у Elite (1984)

Інші відомі ранні приклади включають гру 1985 року Rescue on Fractalus, яка використовувала фрактали для процедурної генерації в реальному часі скелястих гір інопланетної планети, а також River Raid (1982 рік), гру від Activision, що використовувала послідовність псевдовипадкових чисел, згенеровану лінійним зсувом зворотного зв'язку (linear feedback shift register), для створення скролінг-лабіринту з перешкодами.

Хоча сучасні комп'ютерні ігри не мають таких обмежень пам'яті та апаратного забезпечення, як ігри минулих років, використання процедурної генерації часто застосовується для створення випадкових ігор, карт, рівнів, персонажів та інших елементів, які є унікальними при кожному проходженні гри.

У 2004 році була випущена комп'ютерна шутер-гра від першої особи «.kkrieger», розроблена німецькою демо-групою. Вона повністю вміщується в 96-кілобайтному файлі для Windows і генерує сотні мегабайт 3D-даних та текстур під час запуску. За словами одного з програмістів, "це був повний провал з точки зору гри (в основному через те, що ніхто з учасників справді не піклувався про цей аспект)".

Інша гра, RoboBlitz (2006 рік) від Naked Sky, використовувала процедурну генерацію для максимізації контенту в файлі розміром менше 50 МБ, доступному для завантаження через Xbox Live Arcade. Гра Spore 2008 року, розробника Вілла Райта, також використовує процедурний синтез для створення контенту в грі.

Процедурна генерація часто використовується в системах скарбів в іграх, орієнтованих на виконання квестів, таких як action-RPG та масові багатокористувацькі он-лайн рольові ігри (MMO-RPG). Хоча деякі квести можуть мати фіксовані винагороди, інший скарб або лут (loot), наприклад, зброя та броня, може бути згенерований для гравця на основі рівня його персонажа, рівня квесту, його результатів у квесті та інших випадкових факторів. Це часто призводить до того, що лут має таку властивість як «рідкість», що відображає коли система процедурної генерації створила предмет з кращими, ніж середні, характеристиками. Наприклад, серія ігор

Borderlands базується на системі процедурної генерації, яка може створювати понад мільйон унікальних гвинтівок та іншого спорядження. Така висока варіативність і зав’язка геймплею на системі луту, створила новий піджанр ігор, який отримав відповідну назву looter-shooter.

Багато відкритих світі або ігор на виживання процедурно генерують ігровий світ із випадкового насіння або одного, наданого гравцем, так що кожне проходження гри є унікальним. Ці системи генерації створюють численні біоми на основі пікселів або вокселів, з розподілом ресурсів, об'єктів і істот. Гравець часто має змогу коригувати деякі параметри генерації, такі як визначення кількості водних просторів у світі. Прикладом таких ігор є Dwarf Fortress, Minecraft та Vintage Story. На Рис. 1.7 зображено кадр з гри Minecraft, в яка здатна генерувати безмежні унікальні ландшафти та печери засновані на воксельних блоках.

Рис. 1.7 – Процедурний воксельний ландшафт у грі Minecraft.

Процедурна генерація також використовується в іграх про космічні дослідження та торгівлю. Elite: Dangerous (2014 рік), використовуючи 400 мільярдів відомих зірок нашої галактики, застосовує процедурну генерацію для моделювання планет у цих сонячних системах. Подібним чином Star Citizen (в розробці з 2012 року) використовує цю технологію для створення безперервно завантажуваних планет у своєму вручну створеному всесвіті. Outerra Anteworld — відеогра, що знаходиться на стадії розробки, яка використовує процедурну

генерацію та реальні дані для створення віртуальної копії планети Земля в справжньому масштабі.

Гра No Man's Sky 2016 року, завдяки процедурній генерації, стала найбільшою відеогрою в історії, яка охоплює всесвіт з 18 квінтильйонами планет у різних галактиках, які можна досліджувати як в польоті, так і пішки. Кожна планета має унікальний ландшафт, погоду, флору і фауну, а також ряд космічних інопланетних видів. Оскільки одна і та ж контентна інформація зберігається в однакових місцях для всіх гравців (завдяки єдиному «випадковому насінню» для їх детерміністичного ігрового рушія), гравці можуть зустрічатися та ділитися своїми відкриттями.

1.8 Процедурна генерація в кіноіндустрії

Як і в відеоіграх, процедурна генерація часто використовується в кінематографі для швидкого створення візуально цікавих і точних просторів. Це застосовується в різноманітних сферах.

Одне з таких застосувань називається «неідеальна фабрика» (imperfect factory), де художники можуть швидко генерувати багато схожих об'єктів. Це враховує факт, що в реальному житті жоден об'єкт не є абсолютно ідентичним іншому. Наприклад, художник може змоделювати товар для полиці в магазині, а потім створити неідеальну фабрику для генерування багатьох схожих об'єктів, щоб заповнити цю полицю.

MASSIVE — це висококласне програмне забезпечення для комп'ютерної анімації та штучного інтелекту, яке використовується для створення візуальних ефектів, пов'язаних з натовпами, для фільмів і телебачення. Воно було розроблено для автоматичного створення армій з сотень тисяч солдатів у фільмах «Володар перстнів» Пітера Джексона.

Когерентний шум, який в цьому контексті, означає функцію, що генерує плавну псевдовипадковість у n-вимірах, може бути надзвичайно важливим для процедурного процесу в кінематографії. Симплексний шум або Simplex noise, часто є швидшим і має менше артефактів, хоча також може використовуватись і старіша функція – шум Перліна.

В поєднанні з сучасними технологіями в 3D графіці та анімації,

використання процедурної генерації можна вивести на небачений рівень. Так, наприклад, для фільму у жанрі фантастики «Людина-мураха та Оса: Квантоманія» (Ant-Man and The Wasp: Quantumania), що вийшов на великі екрани у 2023 році, компанія з віртуального виробництва та створення візуальних ефектів Pixomondo, використала технологію Bifrost для графічного пакету Autodesk Maya, щоб створити так званий «Фантастичний ліс» у вигаданому квантовому вимірі (див Рис. 1.8).

Рис. 1.8 – Кадр з фільму «Людина-мураха та Оса: Квантоманія»

Bifrost – це середовище візуального програмування в Maya, яке використовується для створення процедурних симуляцій і ефектів, включаючи дим, вогонь, пісок, сніг або, у випадку з Людиною-мурахою, процедурне розсіювання на місцевості.

VFX-команда досліджувала процедурні способи заповнення лісу деревами та рослинністю, і Bifrost дозволив налаштувати параметри для створення 100 дерев, які виглядали однаково за характеристиками, але відрізнялися висотою та товщиною. Коли можна заповнити ліс, не моделюючи кожен окремий об'єкт вручну, це значно економить час. Врешті-решт, вони використали Bifrost для розкиду всього, що доповнювало сцену «Фантастичного лісу». Кожна сцена мала різні розкидані елементи: від сіток, що з'єднували дерева і землю, до грибів на лісовому підлозі, які потім

додавалися до макету (див Рис. 1.9).

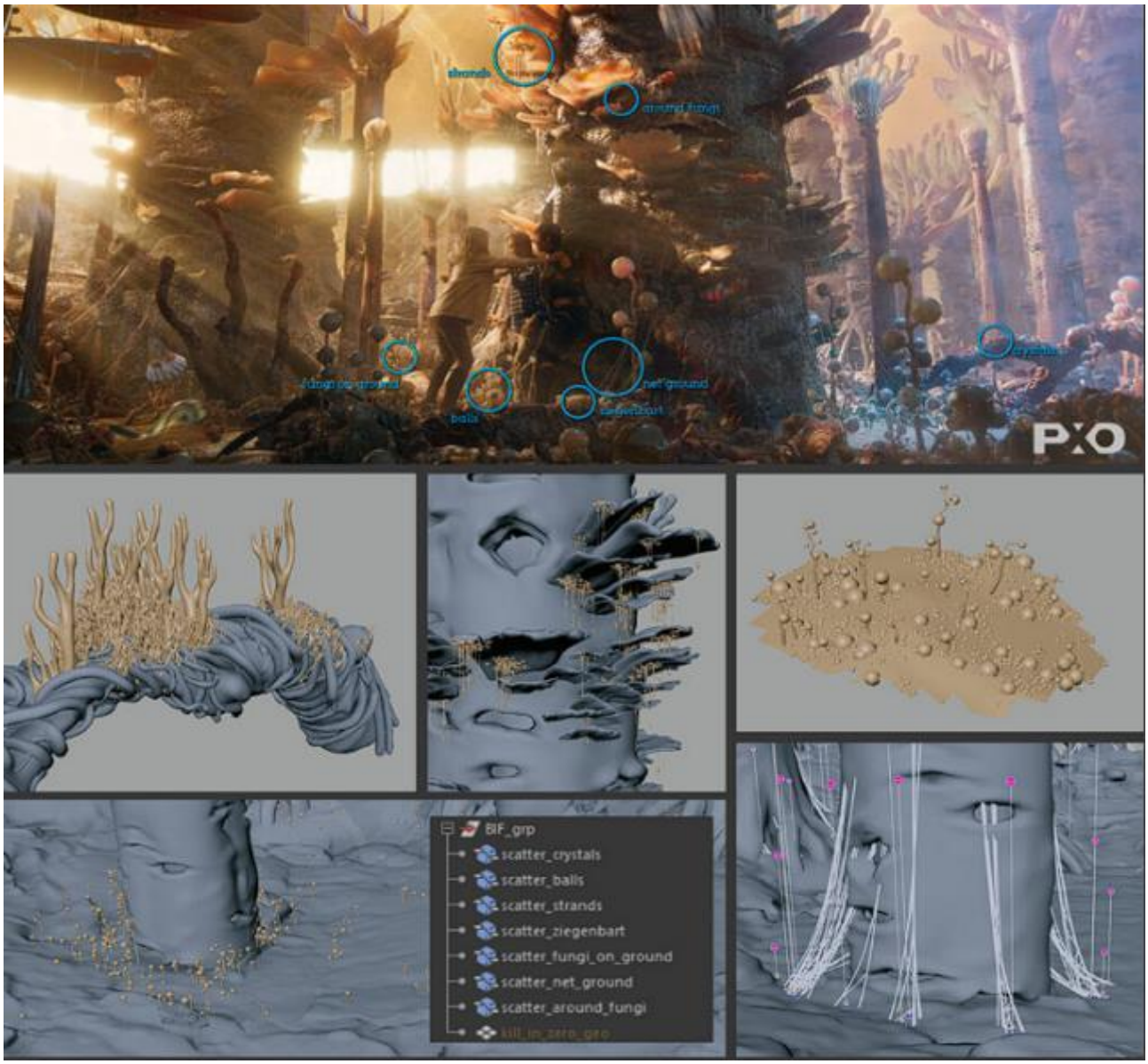


Рис. 1.9 – Процедурно розміщені у сцені елементи декорацій

Весь процес в Maya був організований таким чином, що команда розробила робочий процес з Bifrost і створила інтерфейс, який дозволяв імпортувати макет, геометрію та всі графи одним кліком. Потім можна було вивести найважливіші параметри та визначити, скільки грибів потрібно, чи яка сила тяжіння впливає на нитки. Робота була розподілена між художниками, що значно спростило процес, оскільки їм не потрібно було заглиблюватися в самі графи — достатньо було маніпулювати параметрами. Завдяки одній кнопці команда могла публікувати заповнені сцени і передавати їх для освітлення. Такий робочий процес дозволив швидко виробляти кадри, які виглядають послідовно, але мають різні точки інтересу. Bifrost дав можливість художникам

швидко створювати сотні активів зі спільними характеристиками без моделювання кожного окремого активу вручну або копання в самому графіку (Рис. 1.10).

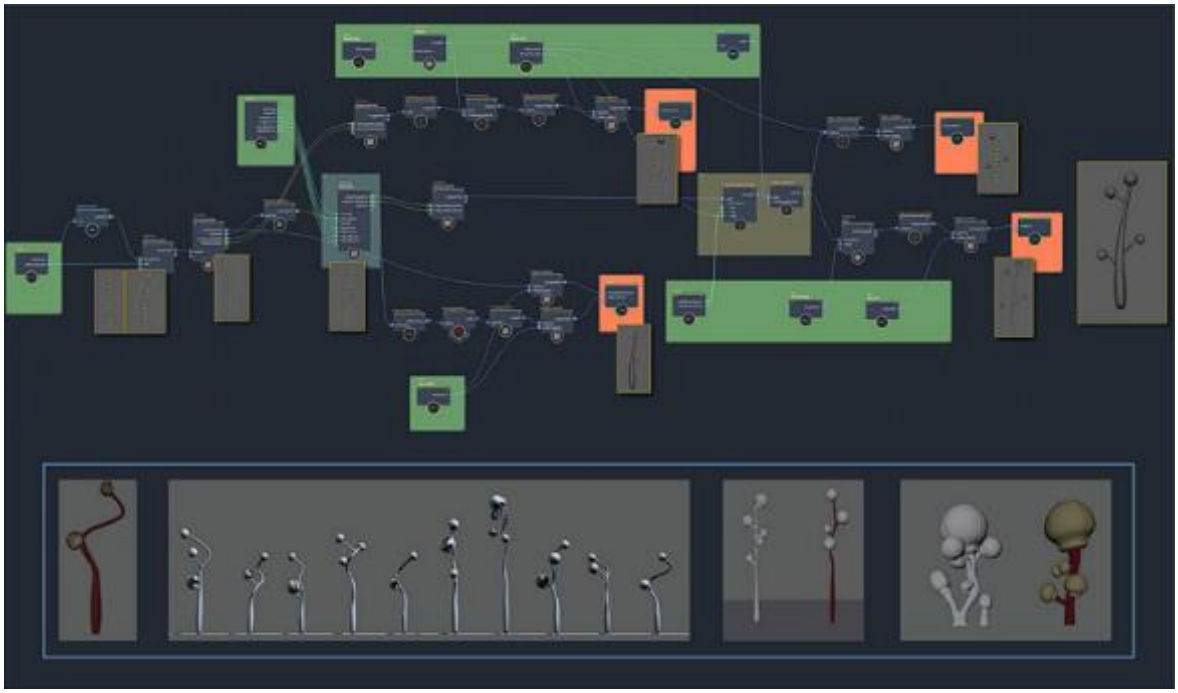


Рис. 1.10 – Граф що задає правила для процедурного розміщення графічних об’єктів у сцені

Модульний підхід у Bifrost значно полегшує роботу, дозволяючи команді швидко і ефективно реагувати на вказівки супервізора. Наприклад, якщо потрібно додати або прибрати об'єкти в сцені, команда може швидко налаштувати граф. Bifrost дуже швидкий у попередній візуалізації, відображення буде дуже точним, що дозволяє більше часу витратити на ітерації. Хоча Bifrost вимагає часу для налаштувань на початку, він дуже гнучкий, коли потрібно зробити специфічні корективи та доопрацювання для вдосконалення сцени.

Загалом, Bifrost є великим заощаджувачем часу, оскільки дозволяє створювати маленькі компаунди, і якщо їх потрібно використати в іншому кадрі чи контексті, їх можна просто взяти і скопіювати в новий граф. Оскільки можливості Bifrost з часом розвиваються, можна повторно використовувати елементи з попередніх проєктів, не винаходячи колесо знову, а просто покращувати його новими функціями.

Висновки до розділу 1.

У цьому розділі було проаналізовано походження та процеси виникнення процедурної генерації, основні поняття та поширеність даної технології у різних сферах.

Процедурна генерація ідейно виникла в настільних іграх «Dungeons & Dragons» у форматі кидання грального кубиків для створення випадкових ситуацій і згодом оцифрувалась у вигляді опублікованих детермінованих таблиць для більш структурованої побудови грального мап.

Одні з перших комп'ютерних ігор, які були натхненні настільними рольовими іграми, окрім сюжетних ідей також перейняли принцип випадкових грального рівнів, або так званих «підземель». Першою грою, що застосувала даний принцип процедурної генерації була Rogue 1980 року. Вона стала дуже популярною, завдяки чому почали з'являтись інші подібні ігри, що утворили піджанр комп'ютерних рольових roguelike.

Визначною особливістю жанру roguelike стали процедурна генерація рівнів та перманентна смерть. Згодом, було визначено 13 пунктів що характеризували жанр, а сам список був названий Берлінським тлумаченням roguelike. Багато хто був не згоднен з таким обмеженням жанру і характерною високою складністю roguelike ігор, тому почали з'являтись ігри, так звані roguelike-подібні або roguelike-like і з годом закріпились як окремий жанр, більш відомий як rogue-lite.

Особливістю rogue-lite стало спрощення деяких елементів складності, наприклад, перманентної смерті, натомість додавання збереження прогресу та елементів покращення персонажу, полегшило гру для менш здібних гравці і підвищило популярність жанру та таких продажі ігор, та збільшило популярність процедурного контенту в різних жанрах ігор.

Жанри roguelike та rogue-lite популяризували процедурну генерацію в ігровій індустрії, а вона в свою чергу, зробила великий вклад у розвиток процедурної генерації. Окрім створення унікальних ігрових рівнів, процедурна генерація використовувалась для створення більш якісних та реалістичних текстур, симуляції природного ландшафту, розвиток 3D графіки для ігор та

кіно, наукових досліджень, тощо.

Процедурна генерація стала необхідним інструментом, який полегшив створення великих об’ємів контенту, який раніше робився вручну, з можливістю налаштовувати ступінь унікальності чи детермінованості таких даних.

РОЗДІЛ 2. ОСНОВНІ МЕТОДИ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ

2.1 Шум Перліна

Шум Перліна є фундаментальним інструментом у процедурній генерації завдяки своїй здатності створювати плавні, неперіодичні патерни, які імітують природні явища. Його унікальні властивості забезпечують реалістичність і деталізацію, що робить його ключовим компонентом у графічному рендерингу, створенні віртуальних середовищ та звукових ефектів.

Розроблений Кеном Перліном у 1983 році, він має багато застосувань, найчастіше його реалізують у двох, трьох або чотирьох вимірах, але він може бути визначений для будь-якої кількості вимірів (див Рис. 2.1), наприклад:

- 2D шум: Для текстур поверхонь.
- 3D шум: Для об'ємних структур, таких як хмари або туман.
- 4D шум: Додаючи час четверту координату, можна створювати динамічні ефекти, коливання води або плавний рух частинок у симуляціях.

У процедурній генерації звуків шум Перліна використовується для додавання природних коливань, які нагадують шум вітру, води або навіть звуки в середовищі.

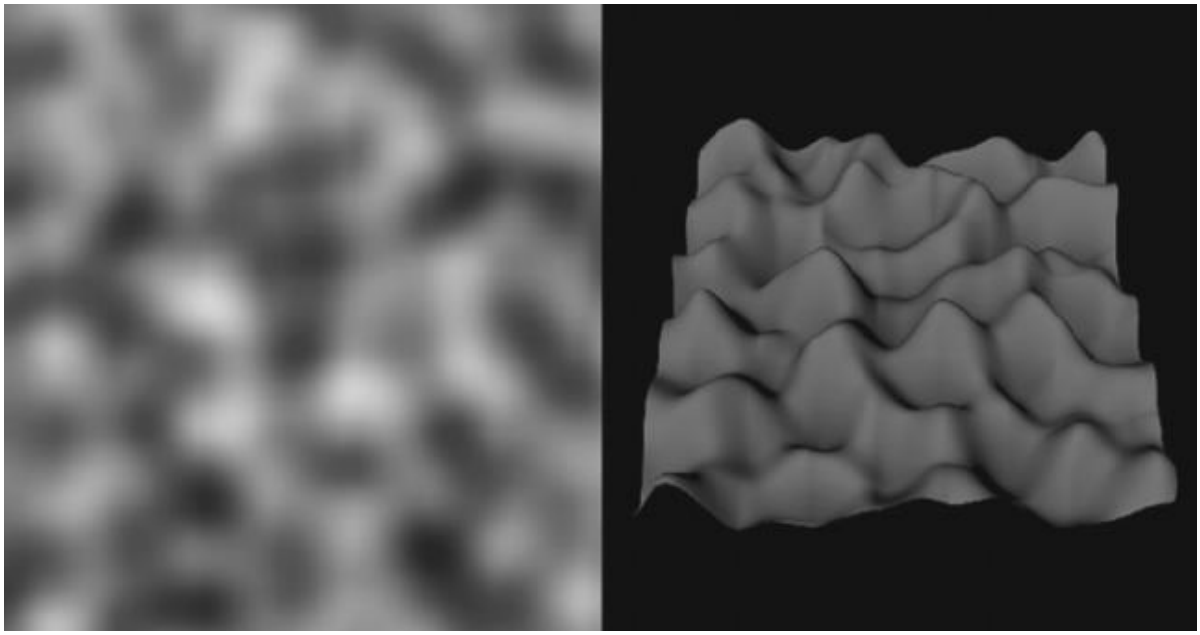


Рис. 2.1 – демонстрація однакових значень шуму Перліна у 2D та 3D

Шум Перліна є процедурним текстурним примітивом і типом градієнтного шуму, який використовується художниками з візуальних ефектів для підвищення реалістичності комп'ютерної графіки. Ця функція має

псевдовипадковий вигляд, але всі її візуальні деталі однакового розміру. Завдяки цій властивості шум Перліна легко контролювати, кілька масштабованих копій шуму можуть бути інтегровані в математичні вирази для створення широкого спектра процедурних текстур.

Також шум Перліна часто застосовується для створення текстур у середовищах з обмеженим обсягом пам'яті, наприклад, у демосценах. Його наступники, такі як фрактальний шум і сиплексний шум, стали практично всюдисущими в графічних процесорах, як у реальному часі, так і для створення процедурних текстур у обчисленнях по за реальним часом.

У відеоіграх шум Перліна часто використовується для процедурної генерації ландшафтів, які виглядають природно (див Рис. 2.2). Успіх цього підходу частково пов'язаний із ієрархічною структурою шуму Перліна, яка імітує природні ієрархічні структури. Ця властивість також знайшла застосування в екологічних наукових дослідженнях, де моделювання природних систем і текстур є важливим аспектом.

Рис. 2.2 – Природний ландшафт створений з допомогою шуму Перліна

Типова реалізація шуму Перліна включає три основні кроки:

- Визначення решітки випадкових градієнтних векторів
- Обчислення скалярного добутку між градієнтними векторами та їх зміщеннями
- Інтерполяція між значеннями

Спочатку простір розбивається на регулярну сітку (решітку), у кожному

вузлі якої генерується випадковий градієнтний вектор. Ці вектори визначають напрямок і інтенсивність зміни значення шуму поблизу кожного вузла (див Рис. 2.3)

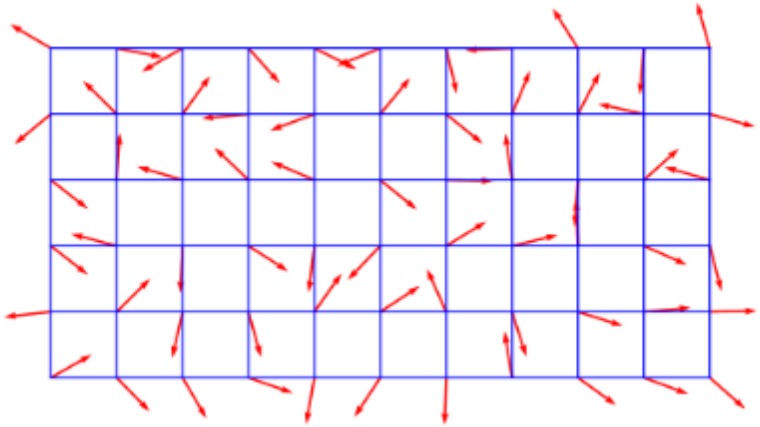


Рис. 2.3 – Двовимірна сітка градієнтних векторів

Для кожної точки простору обчислюється вектор зміщення від цієї точки до найближчих вузлів решітки. Далі для кожного вузла обчислюється скалярний добуток між градієнтним вектором вузла та відповідним вектором зміщення. Це дозволяє визначити внесок кожного вузла в загальне значення шуму в даній точці (див Рис. 2.4).

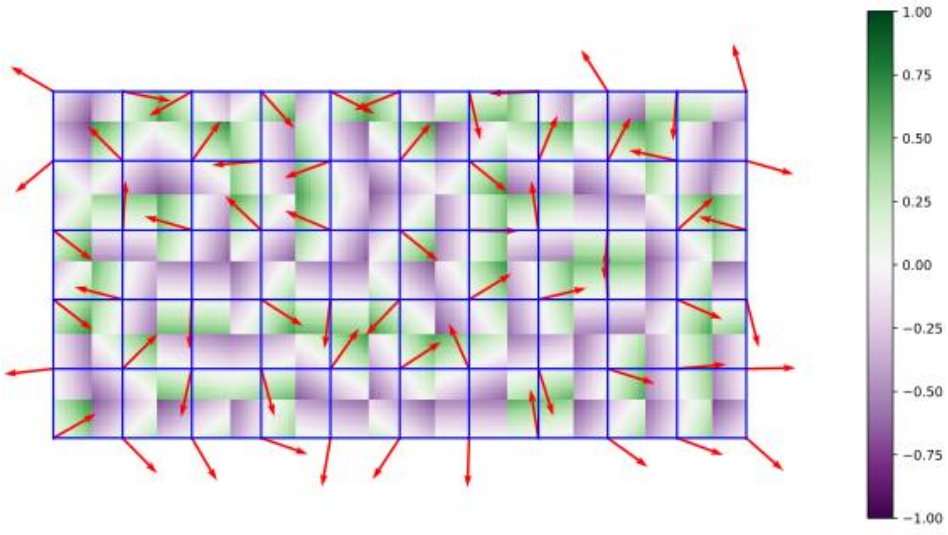


Рис. 2.4 – Скалярний добуток кожної точки сітки.

Отримані значення скалярних добутків інтерполюються для створення плавного переходу між ними. Інтерполяція виконується за допомогою функції, яка має нульову першу похідну (а іноді й другу) у вузлах решітки. Завдяки цьому, поблизу вузлів вихідне значення шуму наближається до скалярного

добутку градієнтного вектора вузла і вектора зміщення до нього. Це доводить, що функція шуму проходить через нуль у кожному вузлі решітки (див Рис. 2.5), що є характерною рисою шуму Перліна.

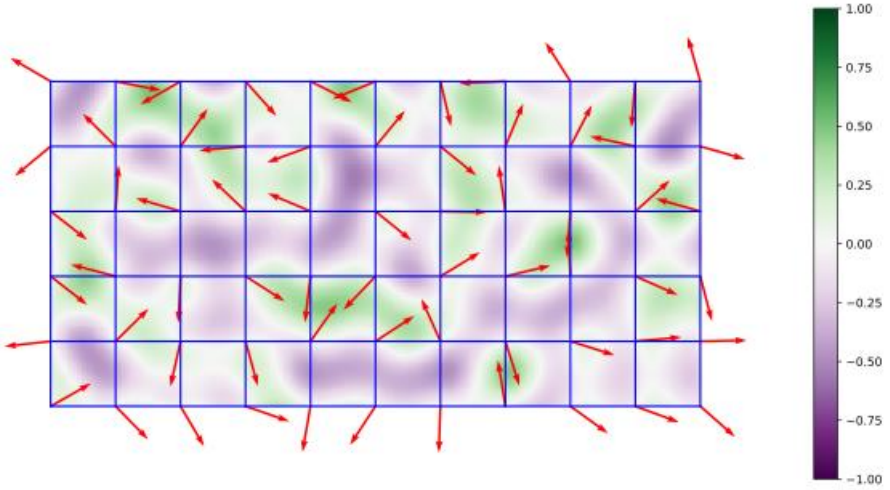


Рис 2.5 – Інтерпольований результат

Програмну імплементацію методу наведено в Додатку А, зокрема двовимірну реалізацію класичного шуму Перліна, написану мовою C. Оригінальна ж референсна реалізація, створена Перліном, мала значні відмінності. Вона використовує тривимірний підхід з інтерполяцією між 8 вершинами куба, на відміну від 4 кутів квадрата в цій двовимірній реалізації.

Напрям випадкових градієнтів визначається через перестановку бітів цілих координат вузлів. Цей метод значно швидший, ніж перестановка шляхом високочастотної інтерференції обертів цілих координат вузлів, які потім комбінуються та знову обертаються на високій частоті за допомогою множення. Однак такі оберти не рівномірно розподілені.

Метод Перліна розділяє простір цілих чисел на куби розміром 256×256×256 і використовує випадкову перестановку цих кубів для їх перемішування. Потім кожному вузлу куба призначається один із дванадцяти напрямків до сусідніх, непереставлених кубів у просторовій решітці 4×4×4. Це вимагає лише цілочисельних операцій, але забезпечує рівномірний розподіл напрямків.

Функція інтерполяції є згладжувальною п’ятого ступеня (smootherstep), яка забезпечує нульові другі похідні на межах (клампінгу). Це кращий варіант, ніж базова лінійна інтерполяція, оскільки усуває видимі артефакти, особливо

уздовж вершин або діагоналей, що з'єднують вузли вибірки. Лінійна інтерполяція може створювати анізотропні артефакти (перетворюючи бажаний "білий шум" у "рожевий шум"). У випадку, якщо шум використовується для генерації текстури твердого кристала, такий кристал не буде повністю чорним і непрозорим, а частково прозорим і матиме кольорові відтінки в певних напрямках спостереження.

Для кожної оцінки функції шуму необхідно обчислити скалярний добуток векторів положення та градієнта в кожному вузлі комірки сітки, що містить точку. Тому складність алгоритму шуму Перліна зростає експоненційно з кількістю вимірів n, тобто $O(2^n)$.

2.2 Симплексний шум

Симплексний шум або «Simplex noise» є результатом n-вимірної функції шуму, яка порівнянна з шумом Перліна ("класичним" шумом), але має менше напрямкових артефактів у вищих вимірах та меншу обчислювальну складність. Кен Перлін розробив цей алгоритм у 2001 році, щоб подолати обмеження своєї класичної функції шуму, особливо у вищих вимірах.

Переваги Simplex шуму порівняно з шумом Перліна:

- Нижча обчислювальна складність: має нижчу обчислювальну складність і вимагає меншої кількості множень.
- Масштабованість до вищих вимірів: масштабується до вищих вимірів (4D, 5D) з набагато меншими обчислювальними витратами: складність становить $O(n^2)$ для n вимірів, на відміну від $O(n^{2n})$ для класичного шуму.
- Відсутність помітних спрямованих артефактів: хоча шум, згенерований для різних вимірів, візуально відрізняється (наприклад, 2D шум виглядає по-іншому, ніж 2D зрізи 3D шуму, і виглядає все гірше для вищих вимірів).
- Чіткий і безперервний градієнт майже всюди, який можна обчислити дуже швидко.
- Легко реалізується в апаратному забезпеченні.

Класичний шум інтерполює між градієнтами в оточуючих вузлах гіперґратки (наприклад, північний схід, північний захід, південний схід і

південний захід у 2D), тоді як Simplex шум розділяє простір на симплекси (тобто n-вимірні трикутники). Це зменшує кількість точок даних: гіперкуб у n-вимірах має 2^n вершин, а симплекс у n-вимірах має лише $n + 1$ вершин. У 2D трикутники є рівносторонніми, але у вищих вимірах симплекси лише приблизно регулярні. Наприклад, розбиття простору у 3D для цієї функції відповідає тетрагональному дисфеноїдальному заповненню.

Simplex шум корисний для застосувань у комп'ютерній графіці, де шум зазвичай обчислюється в 2, 3, 4 або, можливо, 5 вимірах. Для вищих вимірів n-сфери навколо вершин n-симплексів не настільки щільно упаковані, що зменшує підтримку функції і робить її нульовою в великих ділянках простору.

Реалізація зазвичай включає чотири етапи: перекошування координат, поділ на симплекси, вибір градієнтів і підсумовування ядра.

1.Перекошування координат

Вхідна координата перетворюється за формулою:

$$x' = x + (x + y + \ldots) \cdot F,$$
$$y' = y + (x + y + \ldots) \cdot F,$$
$$\ldots,$$

де:

$$F = \frac{\sqrt{n+1} - 1}{n}.$$

Це має ефект розташування координати на решітці A^*_n, яка є розміщенням вершин гіперкубічної комірки, стиснутої вздовж її головної діагоналі так, що відстань між точками (0,0,…,0) і (1,1,…,1) стає рівною відстані між точками (0,0,…,0) і (1,0,…,0).

Отримана координата (x',y',…) використовується для визначення, в якій похиленій одиничній гіперкубічній комірці лежить вхідна точка (x_b'=floor(x'),y_b'=floor(y'),…) та її внутрішніх координат (x_i'=x'−x_b',y_i'=y'−y_b',…).

2. Поділ на симплекси.

Після цього значення внутрішніх координат (x_i',y_i',…) сортуються за спаданням, щоб визначити, в якому похиленому симплексі точка лежить. Отриманий симплекс утворюється вершинами, які відповідають впорядкованому переходу від (0,0,…,0) до (1,1,…,1), серед яких є $n!$ можливостей, кожна з яких відповідає одній перестановці координат.

Наприклад, точка (0.4,0.5,0.3) лежить у симплексі з вершинами (0,0,0), (0,1,0), (1,1,0), (1,1,1). Координата y_i' є найбільшою, тому вона додається першою, потім x_i', і, нарешті, z_i'.

3. Вибір градієнтів.

Кожна вершина симплексу додається до основної координати гіперкуба і хешується в псевдовипадковий напрям градієнта. Хешування може виконуватися різними способами, але найчастіше використовується таблиця перестановок або схема маніпуляції бітами.

При виборі набору градієнтів слід приділяти увагу мінімізації напрямних артефактів.

4. Підсумовування ядра.

Вклад кожної з n+1 вершин симплексу враховується шляхом підсумовування радіально симетричних ядер, центрованих навколо кожної вершини. Спершу визначається нескошена координата кожної з вершин за формулою:

$$x = x' - (x' + y' + \ldots) \cdot G,$$
$$y = y' - (x' + y' + \ldots) \cdot G,$$
$$\ldots,$$

де:

$$G = \frac{1 - 1/\sqrt{n+1}}{n}.$$

Ця точка віднімається з вхідної координати для отримання нескошеного вектора зміщення.

Цей нескошений вектор використовується для двох цілей:

1. Для обчислення екстрапольованого значення градієнта за допомогою скалярного добутку.

2. Для визначення d^2 – квадрату відстані до точки.

Вклад кожного ядра обчислюється за формулою:

$$\left(\max(0, r^2 - d^2)\right)^4 \cdot (\langle \Delta x, \Delta y, \ldots \rangle \cdot \langle \text{grad } x, \text{grad } y, \ldots \rangle),$$

де r^2 зазвичай дорівнює 0.5 або 0.6: значення 0.5 гарантує відсутність розривів, тоді як 0.6 може підвищити візуальну якість у додатках, де ці розриви не помітні. Значення 0.6 використовувалося в оригінальній реалізації Кена Перліна.

Цікаво, що реалізації алгоритму «Simplex Noise» для 3D і вищих вимірів, зокрема для створення текстурованих зображень, були охоплені патентом США №6,867,776. Термін дії цього патенту закінчився 8 січня 2022 року, що означає, що з цього моменту використання цих методів більше не обмежується патентним правом і є доступним для загального використання. Натомість класичний алгоритм шуму Перліна ніколи не був запатентований.

2.3 Процедурна генерація рівнів

Процедурна генерації карти є ключовою особливістю roguelike ігор. Для жанру, який майже синонімічний із «випадковістю» (в розумних межах), випадково побудовані карти – це найпростіший спосіб широко втілити цей ключовий елемент, оскільки карти впливають на багато аспектів ігрового процесу: від стратегії дослідження і тактичного розташування до розміщення предметів і ворогів.

Випадкові карти дарують нам нескінченну можливість повторного проходження, пропонуючи різні виклики щоразу. Крім того, вони приносять особливе задоволення від подолання завдань, які важче зіпсувати спойлерами, де прогрес гравця вимірюється особистою майстерністю, а не методом спроб і помилок. А той факт, що планування кожної нової карти майже невідоме, додає дослідженням чимало напруги.

Звісно, переваги процедурно згенерованих карт втрачають сенс без

достатньої різноманітності механік і контенту, щоб зробити багаторазове проходження вартим уваги – одна й та сама рутина в стилі «рубанини» набридає. Тому roguelike ігри, які витримують перевірку часом, зазвичай мають глибокий ігровий процес.

Існує велике різноманіття методів генерації карт. Звісно, жоден із них не підходить для всіх цілей, тому більшість розробників зрештою коригують обрані методи, щоб краще відповідати потребам своєї гри. Зрозуміло, можливо також розробити власний метод із нуля, хоча концептуально деякі з поширених підходів — це гарна відправна точка для вивчення цієї теми.

Надалі буде наведено короткий опис найпопулярніших методів процедурної генерації мап для roguelike ігор.

1.Генерація на основі кімнат.

Цей метод забезпечує чітку структуру підземелля, де кімнати є основними об'єктами, а коридори забезпечують їх з'єднання (див Рис. 2.6).

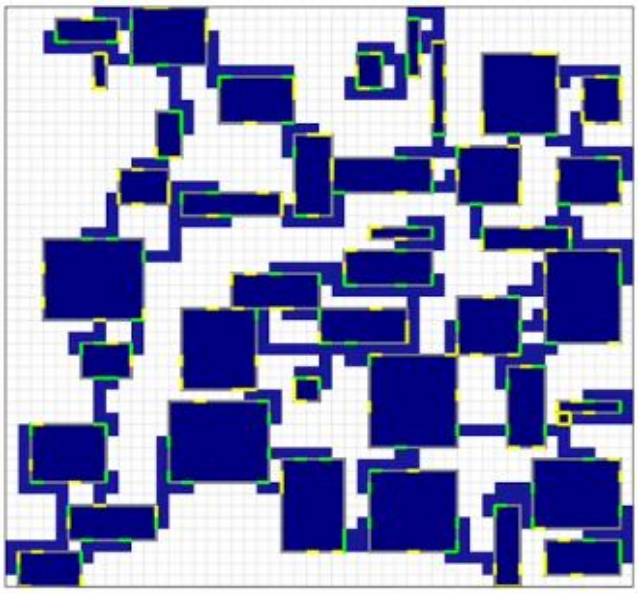


Рис. 2.6 – процедурно побудована мапа рівня на основі кімнат та коридорів

Алгоритм:

1. Випадковим чином визначається кількість кімнат і їх розміри (також можна використовувати заготовлені шаблони кімнат).

2. Кімнати розміщуються у віртуальній сітці підземелля. Якщо кімната перетинається з іншими, її позиція змінюється або вона відкидається.

3. Для з'єднання кімнат використовуються алгоритми для побудови

коридорів між ними. Наприклад: Алгоритм Мінімального кістякового дерева (MST) для зменшення кількості коридорів, Алгоритм A* (або «А зірка») для найкоротшого шляху.

4. Заповнення простору шляхом додавання декоративних елементів, такі як пастки, скарби або монстри.

З переваг даного алгоритму можна виділити легке керування виглядом і структурою, а також досить логічні підземелля. З недоліків, рівень може виглядати занадто «механічно».

Алгоритм клітинних автоматів.

Цей метод ідеально підходить для створення печер або просторів з умовно природним виглядом (див. Рис. 2.7). Алгоритм має певну схожість із поведінкою біологічних систем, таких як розростання колоній бактерій.

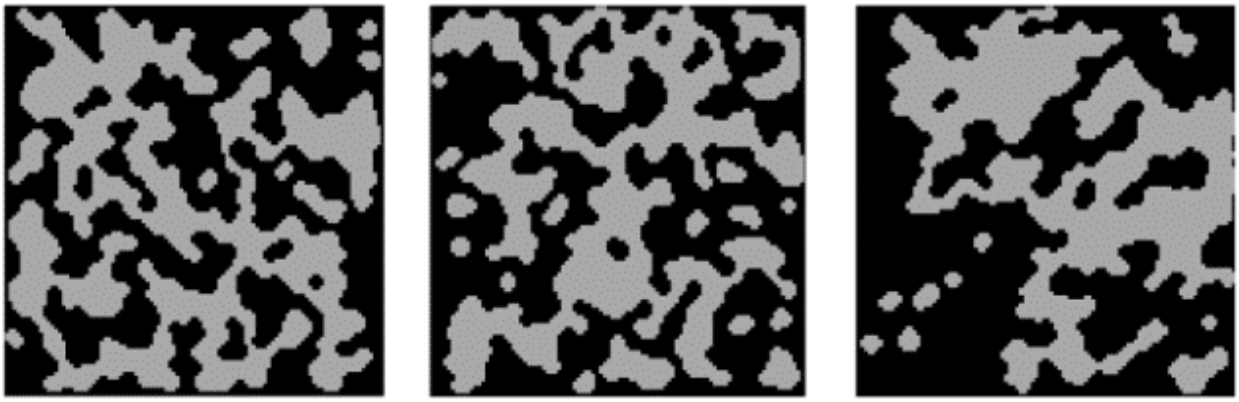


Рис. 2.7 – приклади процедурної генерації підземель з використанням алгоритму клітинних автоматів

Алгоритм:

1. Генерується сітка з випадковим розподілом клітин (стіни і вільний простір). Наприклад, кожна клітина має 40% шанс стати стіною.

2. Кожна клітина змінює свій стан відповідно до кількості сусідів у певному стані. Найчастіше використовуються такі правила:

- Якщо клітина оточена великою кількістю стін (наприклад, ≥4 із 8 сусідів), вона стає стіною.
- Якщо навколо клітини мало стін (наприклад, ≤2 сусідів), вона стає порожнім простором.
- Сусідами зазвичай вважаються клітини в межах 8-ми сусідніх

позицій (верхня, нижня, ліва, права і діагональні)

3. Задаються правила для кожної клітини. Наприклад:

- Якщо навколо клітини ≥4 стіни, вона стає стіною.
- Якщо навколо клітини ≤2 стін, вона стає вільним простором.

4. Застосування правил відбувається кілька разів поспіль (5–10 ітерацій) для досягнення бажаної структури. З кожною ітерацією випадковий "шум" зникає, а структура підземелля стає більш чіткою і схожою на печери або природні простори.

5. Підземелля очищується від ізольованих областей, з'єднуються всі частини підземелля для забезпечення прохідності, наприклад, алгоритмами пошуку шляху A* або BFS для знаходження ізольованих областей.

Основною перевагою алгоритму є природний вигляд, який підходить для органічних підземель. З недоліків можна виділити необхідність тонкого налаштування параметрів, щоб уникнути непрохідних або занадто відкритих просторів.

.Алгоритм поділу простору (BSP - Binary Space Partitioning)

Цей підхід дозволяє створити добре структуровані підземелля з кімнатами і коридорами з допомогою простого рекурсивного поділу простору (див. Рис. 2.8)

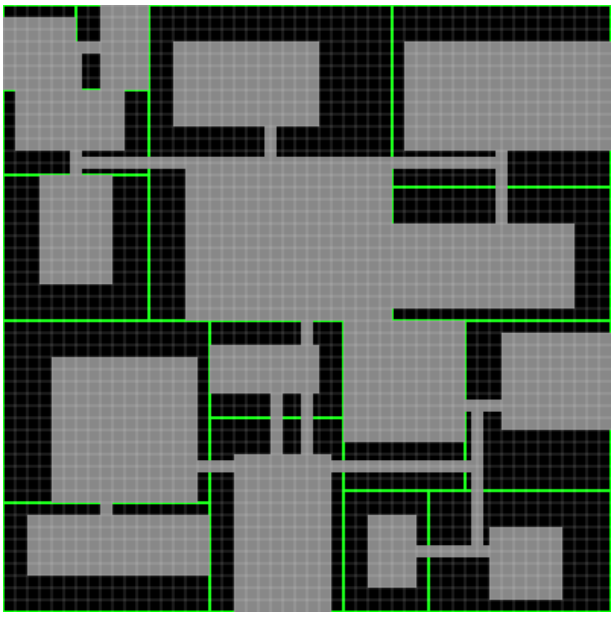


Рис. 2.8 – рівень побудований алгоритмом BSP (лінії поділу схематично показано зеленим)

Алгоритм:

1. Виконується рекурсивний поділ великого простору на дві частини за випадковою лінією (вертикально або горизонтально).

2. Кожна частина ділиться знову, поки її розміри не стануть меншими за заданий мінімум.

3. У кожному підпросторі створюється кімната, яка може займати лише частину області.

4. Кімнати з'єднуються коридорами, що проходять через розділові лінії.

Перевагою алгоритму можна відзначити просту логіку, він добре підходить для великих і симетричних підземель, хоча може виглядати досить штучно

Алгоритм «дріблення стін» (Drunkard's Walk або Random Walk)

Так звана «прогулянка п'яниці» або як ще його часто називають, алгоритм «дріблення стін», створює підземелля через випадковий рух точки в сітці (див Рис. 2.9). Точка «врізається» в новий блок простору, перетворюючи його на частину рівня, після чого розвертається і йде у випадковому напряму, звідки й виникли асоціації з «дрібленням» та «п'яницею»

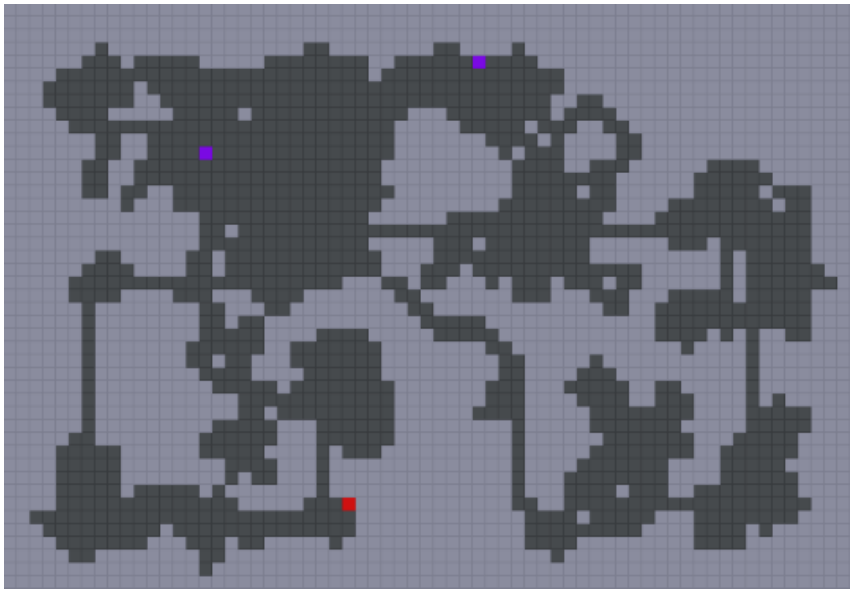


Рис. 2.9 – рівень побудований алгоритмом Drunkard's Walk

Алгоритм:

1. На сітці випадково обирається початкова точка.

2. Ініціалізується рух точки в одному з випадкових напрямків (вгору,

вниз, ліворуч, праворуч).

3. При кожному кроці клітина перетворюється на вільний простір, і випадково обирається напрям наступного кроку (новий або той самий напрям).

4. Процес завершується після досягнення певного відсотка відкритих клітин або заданої кількості кроків.

Перевагами можна назвати легкість реалізації і можливість створення хаотичних, лабіринтоподібних просторів. З недоліків, отриманий рівень може мати погано реалізовану прохідність і невпорядковану структуру.

Алгоритми побудови лабіринту Крускала та Пріма

Це метод створення лабіринтів, який використовує дві популярні стратегії побудови дерев: алгоритм Пріма та алгоритм Крускала. Вони дозволяють побудувати ідеальні лабіринти, де немає замкнутих циклів або недоступних ділянок (див. Рис. 2.10).

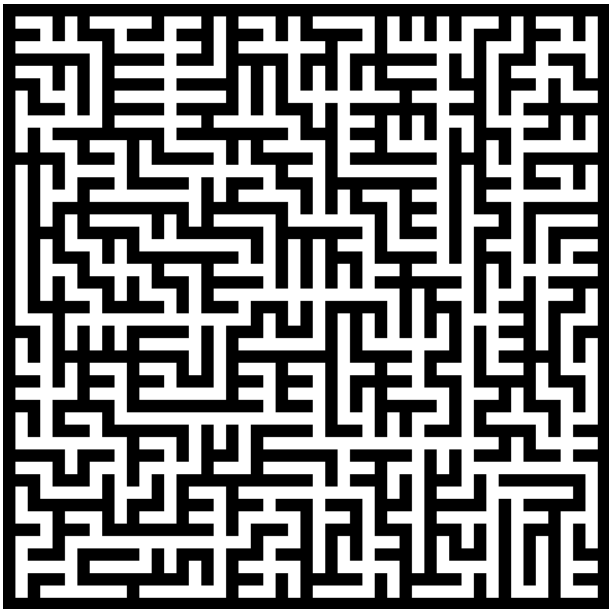


Рис. 2.10 – Приклад лабіритну, згенерованого алгоритмом Пріма

Процес створення лабіринту за допомогою алгоритмів Пріма або Крускала:

1. Спочатку весь простір лабіринту заповнюється стінами. Це можна візуалізувати як сітку клітин, де кожна клітина — це або прохід, або стіна.

2. Одна клітина вибирається як початкова точка, з якої буде починатися побудова лабіринту.

3. Для розширення лабіринту використовується алгоритм Пріма або

Крускала:

- Алгоритм Пріма:
  - Алгоритм Пріма починається з вибору початкової клітини (як правило, це одна з клітин сітки).
  - Потім він вибирає випадкову стіну, яка оточує вже існуючий лабіринт, і руйнує її, тим самим додаючи нову клітину до лабіринту.
  - Цей процес повторюється: нові клітини додаються по черзі, при цьому кожен новий прохід завжди з'єднується з вже існуючими.
  - Таким чином, лабіринт розширюється від центру і поступово заповнює всю сітку.
- Алгоритм Крускала:
  - Алгоритм Крускала працює через побудову дерева пошуку без циклів.
  - Спочатку для кожної клітини лабіринту формуються групи (компоненти), кожна з яких складається з однієї клітини.
  - Потім стіни між клітинами розглядаються по черзі (зазвичай випадковим чином), і стіна руйнується лише в тому випадку, якщо вона з'єднує дві клітини, які належать різним групам.
  - Таким чином, алгоритм поступово з'єднує клітини, поки всі вони не утворять одне дерево без циклів, що і є лабіринтом.

4. Процес генерації лабіринту триває до тих пір, поки всі клітини лабіринту не будуть відвідані (з'єднані між собою). Коли всі клітини з'єднані і немає більше стін, які можна було б зруйнувати, процес вважається завершеним.

Лабіринт, побудований за допомогою Пріма або Крускала, не матиме циклів, що забезпечує, що з кожної точки є єдиний шлях до іншої. Це важливо для створення цікавих, але не надто заплутаних лабіринтів.

Недоліком може бути те, що лабіринти, побудовані за допомогою цих алгоритмів, зазвичай мають дуже структуровану і «щільну» структуру з малим вільним простором.

Граф-орієнтований метод.

Метод базується на використанні графів як абстракції для моделювання структури підземелля. Граф представляє підземелля у вигляді вузлів (кімнат) і ребер (коридорів), що з'єднують ці вузли (див Рис. 2.11).

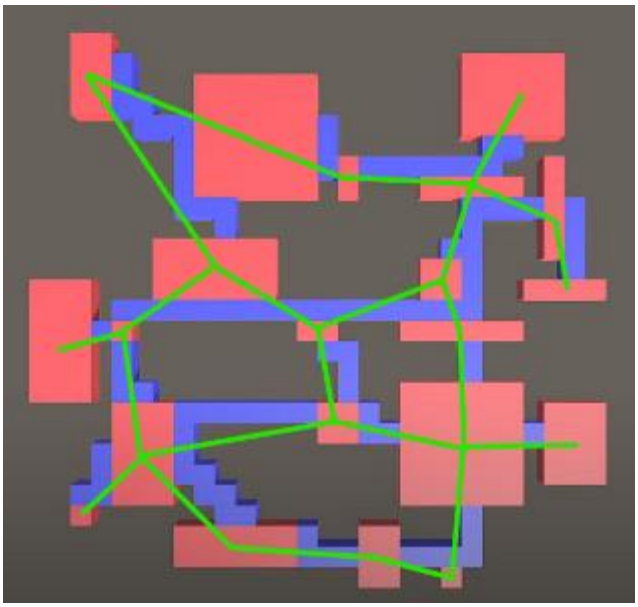


Рис. 2.11 – рівень з кімнатами та коридорами, згенерованими відповідно до графу (граф позначений зеленим)

Алгоритм:

1. Створюється абстрактний граф, де кожен вузол представляє кімнату, а кожне ребро — потенційний коридор між кімнатами. Методи створення графа:

- Повністю зв’язаний граф: Кожна кімната пов'язана з усіма іншими. Це забезпечує максимальну варіативність при наступних етапах.
- Мінімальне кістякове дерево (MST): Алгоритми Крускала або Пріма використовуються для побудови дерева з мінімальною кількістю з'єднань, яке гарантує, що всі кімнати будуть досяжними.
- Додаткові ребра: До MST можна додати деякі випадкові з'єднання для створення кільцевих маршрутів і підвищення складності.

2. Вузли графа розташовуються у фізичному просторі. Це найважчий етап, оскільки необхідно уникнути перетину кімнат або сильного перекриття коридорів. Методи розташування:

- Випадкове розташування: Вузли розташовуються у випадкових координатах.

- Фізичне моделювання: Використовується симуляція сил (відштовхування між кімнатами), щоб уникнути накладань.
- Сіткова система: Простір розбивається на сітку, і вузли графа закріплюються в окремих клітинках.

3. Відбувається побудова коридорів між кімнатами на основі ребер графа. Методи побудови:

- Прямі лінії: Найпростіший варіант, де коридори є прямими лініями між кімнатами.
- Крокування по сітці: Коридори згинаються під прямими кутами, рухаючись по осі X і Y (наприклад, метод «крокуючого потоку»).
- Злиття кімнат: Якщо дві кімнати досить близько, їх можна об'єднати коридором у вигляді тунелю.

Такий алгоритм допомагає легко контролювати кількість вузлів (кімнат), зв’язків (коридорів) і складність шляхів. Можна створювати підземелля різної форми: розгалужені, кільцеві, лінійні тощо. А також є можливість інтегрувати в граф додаткові механізми, наприклад, ключі й двері, створюючи додаткові залежності між кімнатами. Крім того, графи можна розбивати на кілька компонентів, створюючи певні зони підземелля з різними механіками чи тематикою.

Відчутні недоліки полягають у складності розташування, зокрема, уникнути перетинів між кімнатами й коридорами може бути непросто, особливо при використанні складних графів. Також можливі візуальні артефакти, коли занадто геометрично правильні коридори можуть виглядати неприродно. А без спеціальних правил, кімнати можуть бути однаковими за розміром і типом, що зменшує різноманітність підземелля.

Але загалом граф-орієнтовані методи забезпечують чудовий баланс між контролем і випадковістю, що робить їх ідеальним вибором для ігор із процедурною генерацією.

2.4 Wave Function Collapse

Алгоритм Wave Function Collapse (WFC) є потужним інструментом для

процедурної генерації контенту в геймдеві, особливо для створення рівнів, текстур, архітектурних структур або навіть логічних головоломок.

Натхненний квантовою фізикою, WFC походить із квантової механіки, де «колапс хвильової функції» означає перехід від стану суперпозиції (де можливі всі стани) до одного конкретного стану після вимірювання. У WFC цей принцип використовується для поступового звуження можливостей.

Головна ідея алгоритму:

1. Простір розбивається на дискретні клітинки (наприклад, пікселі, блоки, тайли (плитки)).

2. Кожна клітинка на початку має набір можливих станів (наприклад, тайли, які можуть бути розміщені в цій клітинці).

3. Алгоритм поступово «колапсує» клітинки, обираючи для них конкретний стан, враховуючи суміжність із сусідніми клітинками.

Основна мета: генерувати структури, що задовольняють задані правила суміжності, зберігаючи логіку, візуальну гармонію або інші обмеження.

Підготовка набору тайлів або патернів

Заздалегідь визначаються всі можливі стани (наприклад, тайли підлоги, стіни, двері тощо). Для кожного стану описуються правила суміжності (які тайли можуть бути сусідами цього тайла). На Рис. 2.12 зображено набір тайлів для генерації, тут присутні всі варіанти тайлів, які будуть використовуватись в процесі генерації.

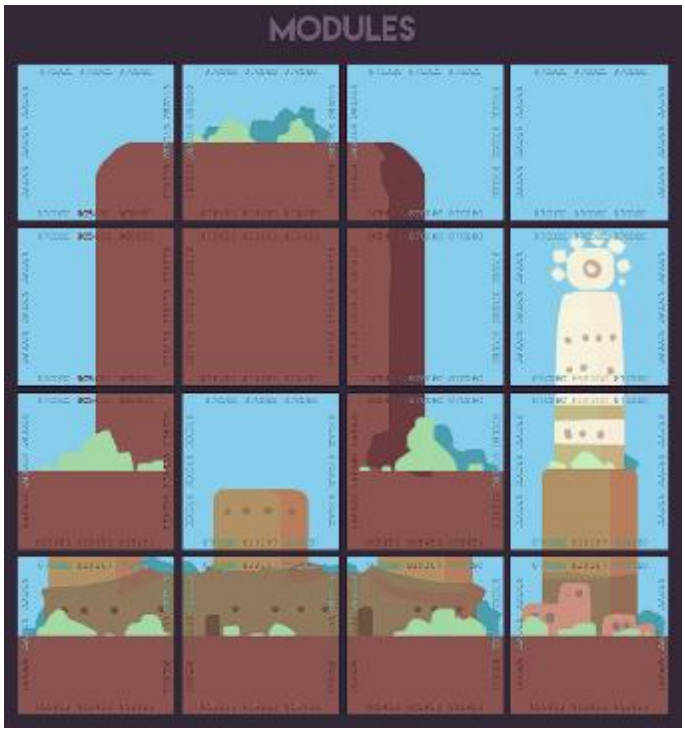


Рис. 2.13 – Повний набір тайлів для генерації

В конкретному випадку, правила суміжності даних тайлів зав’язані на кольорах, а точніше на хеш-кодах кольорів, які прописані по 3 хеш-коди для кожної сторони (верх, низ, лівий і правий боки тайла) і зберігаються для кожного тайла. Наглядно, людське око може розпізнати схожі за кольорами картинки, і скласти «пазл», комп’ютер, в свою чергу, потребує, аби йому прописали ці взаємності.

Ініціалізація простору

Усі клітинки генеративної сітки, на початку перебувають у стані суперпозиції, тобто можуть приймати будь-який із можливих станів. Умовно це можна зобразити, наче кожна клітинка генеративної сітки заповнюється повним набором усіх можливих значень тайлів з переліку (див Рис. 2.13)

Рис. 2.13 – Генеративна сітка, кожна клітинка містить весь набір тайлів

Вибір клітинки з найменшою ентропією.

Ентропія визначається кількістю можливих станів клітинки. Алгоритм вибирає клітинку, яка має найменше число варіантів, і колапсує її до одного стану, тобто обирає для неї один тайл з усього набору, а інші видаляє з поточної клітинки. Оскільки на початку алгоритму, всі клітинки мають однакову ентропію, для першої ітерації вибирається випадкова або задана клітинка.

Пропагування обмежень.

Після колапсу алгоритм оновлює сусідні клітинки, видаляючи з їхніх можливих станів ті, що порушують правила суміжності. Цей процес повторюється, поки всі клітинки не будуть визначені. На Рис. 2.14 зображений результат першої ітерації, коли одна клітинка отримала єдине значення і «нав’язала» нові правила суміжності сусіднім клітинкам.

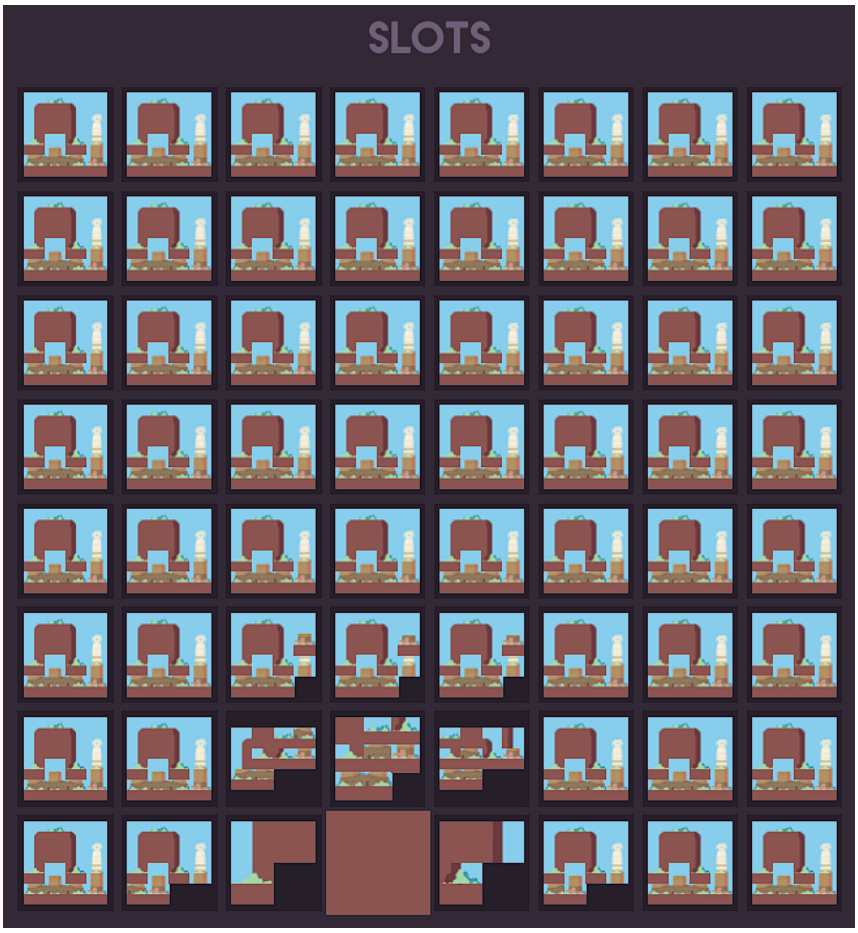


Рис. 2.14 – Результат першої ітерації

Іноді виникають конфлікти суміжності, якщо не залишилося допустимих станів для певної клітинки, алгоритм може почати заново або виконати частковий «відкат» до моменту перед конфліктом і спробувати інший вибір.

Результат

Коли всі клітинки успішно колапсують, отримавши таким чином тільки одне можливе значення з попереднього переліку усіх доступних, генеративна сітка складеться в гармонійну картинку з правильними переходами відповідно до правил суміжності. На Рис. 2.15 можна відслідкувати процес, від набору можливих значень, проміжного етапу та результату.

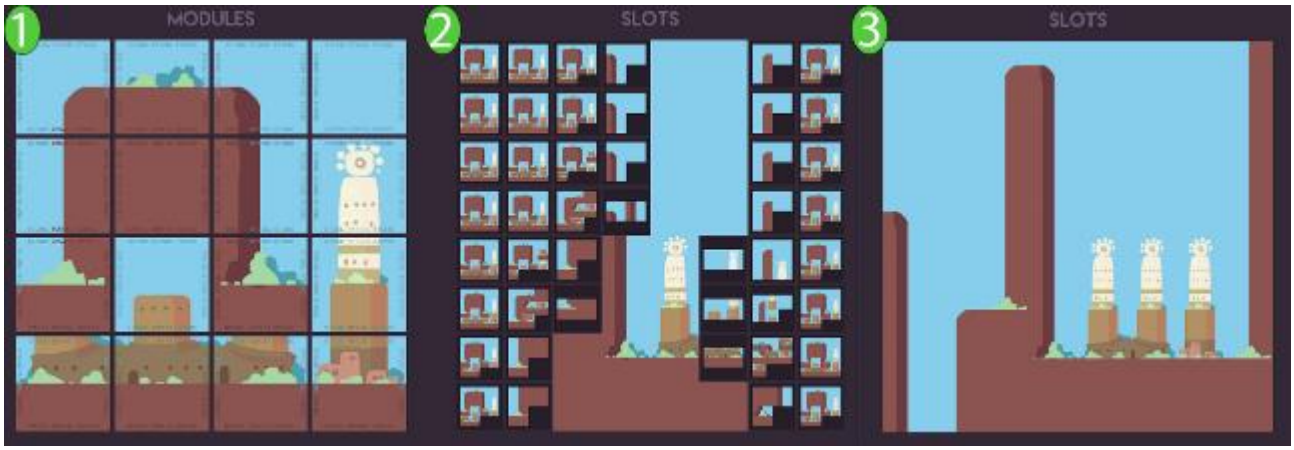


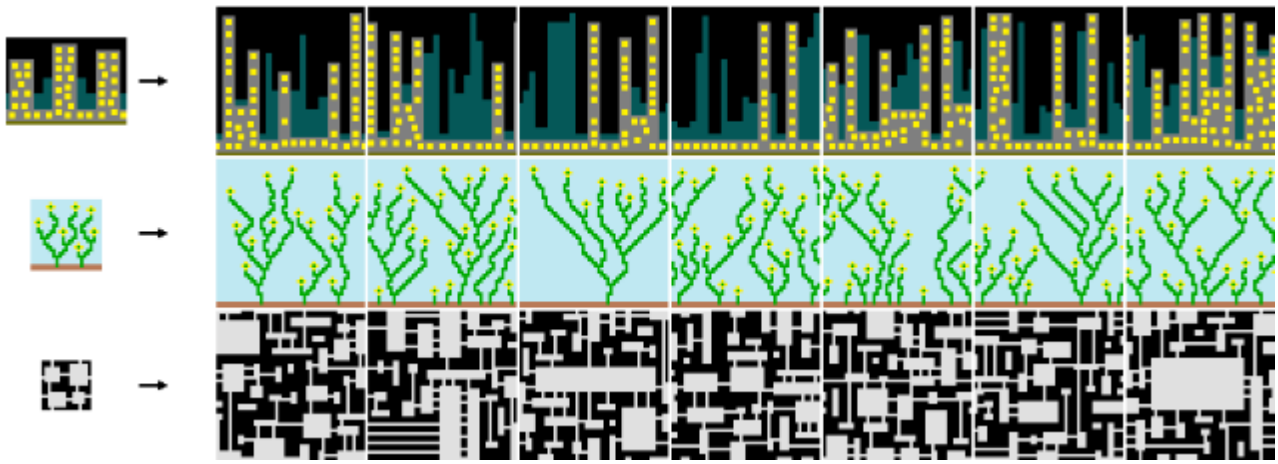
Рис. 2.15 – 1-набір можливих тайлів, 2-генеративна сітка після кількох ітерацій, 3-генеративна сітка після завершення алгоритму.

Даний приклад описує роботу WFC алгоритму на базі тайлів, тобто плиток

Типи WFC-алгоритмів

Вище була описана тайл-орієнтована версія (Tile-based) алгоритму, для якої простір розглядається як сітка і для генерації використовується набір тайлів із правилами їхньої суміжності. Існує також оверлап-орієнтована версія (Overlapping model), для неї простір заповнюється патернами, що перекриваються, а вихідні патерни генеруються з прикладного зображення або шаблона.

Для такої моделі, в якості шаблонів можна використовувати прості растрові зображення (див Рис. 2.16). Замість прослідковувати зв’язки між тайлами, вона буде будувати правила суміжності для патернів розміром N x N пікселів з зображення і відтворювати їх під час генерації перекриванням



патернів.

Рис. 2.16 – Прості растрові зображення як шаблони і результат їх процедурного відтворення

Така модель створює растрові зображення, локально схожі на вхідне растрове зображення. Локальна схожість означає, що вихідні дані повинні містити лише ті NxN шаблони пікселів, які є у вхідних даних. Розподіл NxN шаблонів у вхідних даних має бути подібним до розподілу NxN шаблонів у достатньо великій кількості вихідних даних. Іншими словами, ймовірність зустріти конкретний шаблон у вихідних даних повинна бути близькою до його частоти у вхідних даних. На Рис. 2.17 зображено приклад NxN шаблонів з типовим значенням N = 3.

Рис. 2.17 – Шаблони 3x3 пікселя які алгоритм виділяє з вхідного зображення

Алгоритм WFC ініціалізує вихідну бітову карту у повністю незафіксованому стані, де кожне значення пікселя є суперпозицією кольорів із вхідної бітової карти. Наприклад, якщо вхідні дані були чорно-білими, то незафіксовані стани зображуються різними відтінками сірого.

Коефіцієнти цих суперпозицій – це дійсні числа, а не комплексні, тому алгоритм не виконує реальних квантово-механічних обчислень, але надихався квантовою механікою. Потім програма переходить до циклу "спостереження-пропагування":

1. На кожному етапі спостереження вибирається NxN-область серед незафіксованих, яка має найменшу ентропію за Шенноном. Стан цієї області "колапсує" в певний стан відповідно до її коефіцієнтів і розподілу NxN шаблонів у вхідних даних.

2. На кожному етапі пропагування нова інформація, отримана в

результаті «колапсу» на попередньому кроці, поширюється через вихідні дані.

3. На кожному кроці кількість ненульових коефіцієнтів зменшується, і в результаті алгоритм завершується, досягаючи повністю зафіксованого стану — хвильова функція «колапсує».

Може статися, що під час пропагування всі коефіцієнти для певного пікселя стануть нульовими. Це означає, що алгоритм зіткнувся з суперечністю і не може продовжити роботу. Проблема визначення, чи дозволяє певна бітова карта інші нетривіальні карти, які задовольняють умову, є NP-складною. Тому неможливо створити швидке рішення, яке завжди завершує роботу. Однак, на практиці алгоритм рідко стикається із суперечностями.

Багато вимірні WFC алгоритми

Алгоритм WFC у вищих вимірах працює точно так само, як і у двовимірному випадку, хоча виникають проблеми з продуктивністю. На Рис. 2.18 зображені воксельні моделі, що були згенеровані з використанням перекривного плиткового моделювання при N=2, із блоками розміром 5×5×5 і 5×5×2, а також додаткових евристик (висота, щільність, кривизна, тощо).

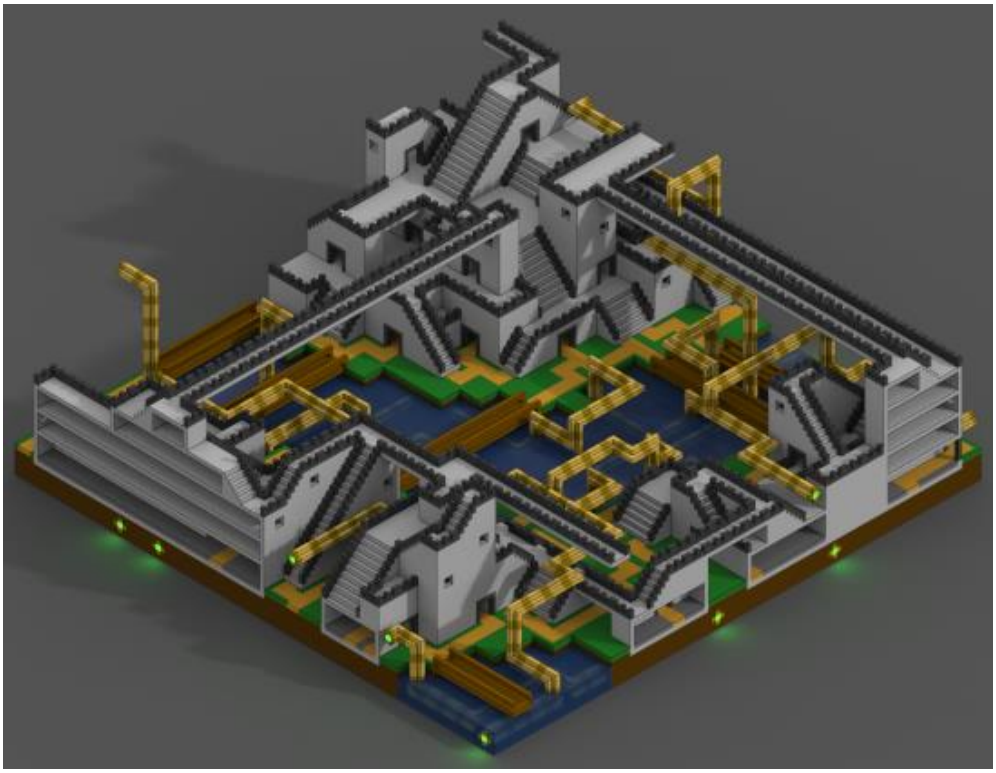


Рис. 2.18 – Результат виконання WFC алгоритму у 3 вимірному просторі.

Висновки до розділу 2

У цьому розділі було детально розглянуто основні напрями та приклади

застосування процедурної генерації.

Зокрема, розглянуто принципи роботи шуму Перліна, фундаментального інструменту в цифровому дизайні, який вплинув на розвиток текстурування, симуляції ландшафтів та обробки даних. Ступінь важливості даної технології підтверджує отримання у 1997 році її автором Кеном Перліном нагороди Оскар «За технічні досягнення» та внесок у індустрію. У 2001 році Перлін розробив модифіковану версію свого алгоритму, яка покращувала моделювання у вищих просторах вимірювань, та отримала назву сиплексний шум (Simplex noise), зокрема через поділ простору на сиплекси.

Також було розглянуто основні методи процедурної генерації рівнів, які розвинулись завдяки популярності жанрів ігор Roguelike та Rogue-lite. Серед них:

- Генерація на основі кімнат
- Алгоритм клітинних автоматів
- Алгоритм поділу простору (BSP)
- Алгоритм «дріблення стін» (Drunkard’s Walk)
- Алгоритми побудови лабіринту Крускала та Пріма
- Граф-орієнтований метод

Було розглянуто більш просунуті алгоритми процедурної генерації, зокрема, моделі алгоритму Wave Function Collapse (WFC), який заснований на ідеях суперпозиції з квантової механіки, хоч і не виконує квантових обчислень, але натхненний саме дослідженнями «колапсу хвильової функції». WFC приймає вхідні дані у вигляді набору всіх можливих станів для кожної виділеної області простору, зберігає правила суміжності для кожного стану. Кожній клітинці впорядкованого простору надає стан суперпозиції, тобто кожна клітинка може містити всі можливі стани. Надалі, шляхом пошуку найменшої ентропії в клітинках, обирає клітинку, яка колапсує до одного можливого значення стану, при цьому сусіднім клітинкам пропагується суміжність з отриманим значенням, несуміжні варіанти видаляються з можливих. Таким чином, послідовно, в кожній клітинці зменшуватиметься кількість можливих станів, доки вони всі не колапсують, завершивши

генерацію. Функція здатна виконуватись в 2 вимірах на базі плиток або патернів растрових зображень і у вищих вимірах, зокрема в 3 вимірному просторі на основі воксельних шаблонів даних.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Програмні засоби

Для виконання програми буде використовуватись ігровий рушій Unity, який підтримує скрипти мовою C#. Unity – потужний інструмент для розробки ігор, який широко використовується для реалізації процедурної генерації контенту. Завдяки своїй гнучкості, підтримці C#, бібліотекам і готовим плагінам (plugins), Unity дозволяє створювати складні ігрові світи, рівні, персонажів і навіть механіки, що генеруються динамічно.

Розглянемо поняття та характеристики ігрового рушія Unity .

Unity має вбудовану функцію Mathf.PerlinNoise, яка дозволяє застосовувати алгоритм шуму Перліна та з його допомогою створювати гладкі випадкові значення для генерації ландшафтів, текстур або інших візуальних елементів. Зокрема, для генерації і ландшафтів і всього пов’язаного з ними існує базовий пакет Terrain tools, який інтегрований в рушій.

Процедурна генерація в Unity може бути реалізована через скрипти C#, плагіни з Unity Asset Store і сторонні бібліотеки. Все це дозволяє виконувати основні застосування процедурної генерації в іграх:

- Рівні та підземелля: автоматична побудова рівнів, лабіринтів, відкритих світів або підземель.
- Ландшафти: створення гір, річок, пустель, лісів із шумами та сплайн-моделями.
- Об'єкти: генерація дерев, будівель, ворогів, перешкод.
- Контент гри: динамічне створення квестів, завдань, історій або NPC.
- Моделі: створення текстур, матеріалів, навіть 3D-об'єктів під час гри.

Unity пропонує безліч плагінів для процедурної генерації, які можна знайти в Unity Asset Store. Вони допомагають швидко створювати різноманітний контент. перелік найпопулярніших плагінів для процедурної генерації.

Огляд найпопулярніших плагінів для процедурної генерації.

1. Dungeon Architect.

Інструмент для процедурної генерації підземель, рівнів і лабіринтів. Використовується для побудови випадкових підземель із різними кімнатами та проходами.

Особливості:

- Гнучкий дизайн на основі правил (blueprints).
- Підтримка як 2D, так і 3D контенту.
- Генерація кімнат, коридорів, дверей і об'єктів.
- Візуальний редактор для налаштування правил.

2. MapMagic 2.

Плагін для створення процедурних ландшафтів. Часто використовується для генерації великих відкритих світів для RPG або survival-ігор.

Особливості:

- Генерація нескінченних світів у реальному часі.
- Налаштування біомів: ліси, пустелі, гори, озера.
- Інтеграція з Unity Terrain System.
- Підтримка детального редагування об'єктів.

3. Gaia Pro.

Інструмент для створення процедурних ландшафтів. Корисний для творення реалістичних ландшафтів із водою, горами та рослинністю.

Особливості:

- Потужні можливості для генерації середовища.
- Інтеграція зі сторонніми ассетами (assets).
- Легке налаштування текстур, трави, дерев.
- Можливість додавання погодних ефектів.

4. Octave3D Level Design.

Інструмент для процедурної генерації рівнів. Найкраще підходить для генерації міст або великих внутрішніх просторів.

Особливості:

- Інструменти для розміщення об'єктів за шаблонами.
- Створення масивних структур, таких як міста, будівлі.
- Підтримка 3D і 2D проєктів.

5. Terrain Composer 2.

Потужний плагін для генерації ландшафтів і карт висот. Створює мапи із підтримкою кастомізації для будь-якого типу гри.

Особливості:

- Легке створення великих і деталізованих карт.
- Інтеграція з текстурами, деревами та травою.
- Підтримка процедурної генерації текстур.

6. PolyFew

Інструмент для оптимізації 3D-моделей і генерації полігональних об'єктів.

Особливості:

- Зменшення кількості полігонів для процедурно згенерованих моделей.
- Генерація LOD (рівнів деталізації).
- Збереження текстур і UV-координат.
- Оптимізація випадково згенерованих моделей у реальному часі.

7. ProBuilder

Вбудований плагін Unity для моделювання. Застосовується для створення кімнат і коридорів процедурним способом.

Особливості:

- Побудова геометрії процедурним чином.
- Інтеграція зі скриптами для автоматизації процесу.
- Зручне редагування 3D-об'єктів прямо в редакторі.

8. Voxeland

Генератор воксельних світів, дозволяє генерувати світи із можливістю їх модифікації як у Minecraft.

Особливості:

- Створення ландшафтів із вокселів.
- Інтеграція з MapMagic.
- Реальна зміна світу під час гри.

9. Instant Meshes Integration

Інструмент для створення та оптимізації мешів (meshes). Використовується у генерації об'єктів для стратегій або симуляторів.

Особливості:

- Процедурне спрощення сіток.
- Генерація текстур і UV-карт.

10. Archimatix

Інструмент для процедурного моделювання архітектури. Популярний у генерації складних будівель і інфраструктур для ігор.

Особливості:

- Створення будівель, мостів, колон.
- Динамічне налаштування об'єктів у реальному часі.
- Інтеграція з іншими ассетами Unity.

Огдяд C# бібліотеки для Unity.

Крім офіційних плагінів з Unity Asset Store, рушій підтримує інтеграцію звичайних скриптів мовою C#, який в свою чергу надає широкий спектр бібліотек, які можуть бути використані в Unity для процедурної генерації та розробки ігор.

Розглянемо деякі з них.

1.LibNoise

Бібліотека для генерації шумів (Perlin, Simplex, Voronoi та інші). Використовується для генерації висот для 3D-ландшафтів або текстур.

Особливості:

- Відкрита бібліотека для роботи з текстурами і висотами.
- Простий у використанні API.
- Можливість створення процедурних ландшафтів, текстур або інших

шумових даних.

2. FastNoise

Високопродуктивна бібліотека для генерації різних шумів. Застосовується для створення випадкових візерунків для текстур або карт висот.

Особливості:

- Генерація шумів Simplex, Perlin, Cellular, OpenSimplex і інших.
- Висока продуктивність завдяки оптимізованому коду.
- Підходить для процедурної генерації текстур у реальному часі.

3. Delaunay.NET

Бібліотека для побудови діаграм Вороного та тріангуляції Делоне. Часто використовується для генерації карт у roguelike іграх або світах в стилі серії ігор Civilization.

Особливості:

- Простий у використанні інструмент для роботи з точками в просторі.
- Використовується для генерації діаграм Вороного, карт, ландшафтів.

4. ProceduralToolkit

Бібліотека для процедурної генерації геометрії, текстур та інших ресурсів. Виконує генерацію міст або карт із складною топологією.

Особливості:

- Створення доріг, будівель, ландшафтів.
- Підтримка роботи з текстурами і колірними палітрами.
- Великий набір готових функцій для розробки процедурних ігор.

5. Math.NET Numerics

Потужна бібліотека для математичних і статистичних розрахунків. Можна використовувати для генерації складних карт або обчислення фізичних моделей.

Особливості:

- Векторні та матричні обчислення.
- Статистичні методи, які можна застосовувати в процедурній генерації.
- Функції для роботи з шумами та обчисленнями.

6. AForge.NET

Бібліотека для комп'ютерного зору, обробки зображень і штучного інтелекту. Застосовується для генерації текстур для об'єктів у 3D-світі.

Особливості:

- Можливості роботи зі спрайтами або текстурами.
- Генерація шумових текстур і процедурних матеріалів.
- Інтеграція з Unity через .NET.

7. Unity-Mathematics

Бібліотека від Unity для високопродуктивних математичних обчислень. Необхідна для виконання високопродуктивних обчислень для великих процедурно згенерованих світів.

Особливості:

- Підтримка векторів, матриць і кватерніонів.
- Інтеграція з Unity DOTS (Data-Oriented Technology Stack).
- Оптимізовано для процедурної генерації в реальному часі.

7. OpenSimplex2

Оптимізована версія OpenSimplex Noise для процедурної генерації. Застосовується для створення плавних переходів у ландшафтах або текстурах.

Особливості:

- Генерація плавних шумів у 2D, 3D і 4D.
- Висока продуктивність і мінімізація артефактів.

8. Accord.NET

Широкий набір бібліотек для машинного навчання, статистики та обробки даних. Корисний для генерації персонажів або ворогів зі складною поведінкою.

Особливості:

- Створення моделей для розумного процедурного контенту.
- Обробка великої кількості даних для складних алгоритмів.

Ці бібліотеки значно полегшують реалізацію процедурної генерації та дають змогу зосередитися на креативному аспекті розробки ігор. Для їх додавання можна використати інтеграцію через Unity Package Manager, просто додати потрібні бібліотеки в проєкт з переліку запропонованих, для доступу до їх функціоналу. Або можна комбінувати їх з власними алгоритмами, бібліотеки можуть слугувати основою, яку можна модифікувати відповідно до власних потреб. Ну і звісно можливе створення кастомних компонентів, шляхом реалізації скриптів для процедурної генерації, використовуючи можливості бібліотек.

Відкритий доступ до Unity, C# та велика кількість надбудов для них у відкритому доступі, надають гарні можливості для розробників використовувати ці пакети для власних розробок, в тому числі і для нашого проєкту.

3.2 Реалізація програми

Було вирішено розробити програму, яка буде використовувати різні методи процедурної генерації на прикладі roguelike гри, для кращої демонстрації можливостей методів процедурної генерації.

План реалізації було розділено на кілька етапів:

1. Реалізація псевдовипадкового розміщення кімнат.

2. Побудова зв’язків між кімнатами.

3. Побудова ігрових об’єктів рівня відповідно до попередніх процедурних процесів.

4. Розробка процедурних анімацій для персонажу гравця.

Таким чином, в проєкті буде виконано процедурну генерацію різних типів та для різних задач.

Розміщення кімнат

Для випадкового розміщення кімнат, виникла ідея застосувати шум Перліна. Оскільки даний алгоритм оперує матрицями даних (див Рис. 3.1), а також візуалізує їх, було вирішено використати ці дві особливості на користь

проєкту.

```
{ 151, 160, 137,  91,  90,  15, 131,  13, 201,  95,  96,  53, 194, 233,   7, 225,
  140,  36, 103,  30,  69, 142,   8,  99,  37, 240,  21,  10,  23, 190,   6, 148,
  247, 120, 234,  75,   0,  26, 197,  62,  94, 252, 219, 203, 117,  35,  11,  32,
   57, 177,  33,  88, 237, 149,  56,  87, 174,  20, 125, 136, 171, 168,  68, 175,
   74, 165,  71, 134, 139,  48,  27, 166,  77, 146, 158, 231,  83, 111, 229, 122,
   60, 211, 133, 230, 220, 105,  92,  41,  55,  46, 245,  40, 244, 102, 143,  54,
   65,  25,  63, 161,   1, 216,  80,  73, 209,  76, 132, 187, 208,  89,  18, 169,
  200, 196, 135, 130, 116, 188, 159,  86, 164, 100, 109, 198, 173, 186,   3,  64,
   52, 217, 226, 250, 124, 123,   5, 202,  38, 147, 118, 126, 255,  82,  85, 212,
  207, 206,  59, 227,  47,  16,  58,  17, 182, 189,  28,  42, 223, 183, 170, 213,
  119, 248, 152,   2,  44, 154, 163,  70, 221, 153, 101, 155, 167,  43, 172,   9,
  129,  22,  39, 253,  19,  98, 108, 110,  79, 113, 224, 232, 178, 185, 112, 104,
  218, 246,  97, 228, 251,  34, 242, 193, 238, 210, 144,  12, 191, 179, 162, 241,
   81,  51, 145, 235, 249,  14, 239, 107,  49, 192, 214,  31, 181, 199, 106, 157,
  184,  84, 204, 176, 115, 121,  50,  45, 127,   4, 150, 254, 138, 236, 205,  93,
  222, 114,  67,  29,  24,  72, 243, 141, 128, 195,  78,  66, 215,  61, 156, 180 };
```

Рис. 3.1 – Значення вершин функції шуму Перліна у вигляді матриці

Кімнати будуть розміщуватись в точках екстремумів тривмимірної функції шуму Перліна.(див Рис. 3.2)

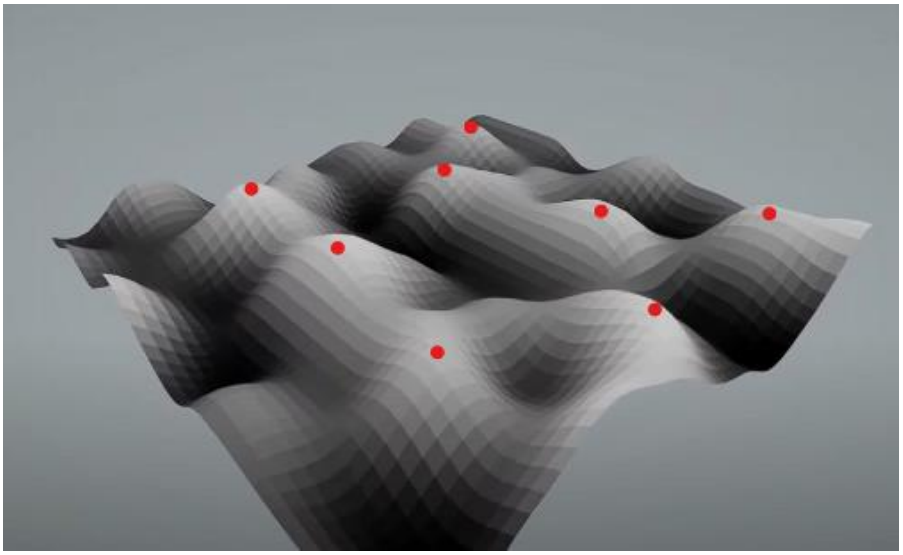


Рис. 3.2 – Вершини ландшафту як точки розміщення кімнат

Це дозволить використовувати координати вершин функції в якості маркерів для розміщення випадкових кімнат. В свою чергу, так можна уникнути перетинань кімнат, яке може виникати при звичайному випадковому розміщенні, оскільки в шумі Перліна між вершинам досить плавний перехід, і широка вибірка значень.

Для виконання цього, перш за все треба створити меш-сітку, і накласти на неї текстуру шуму Перліна. Обидві дії виконуються відповідними скриптами, а результат виглядає наступним чином (див Рис. 3.3)

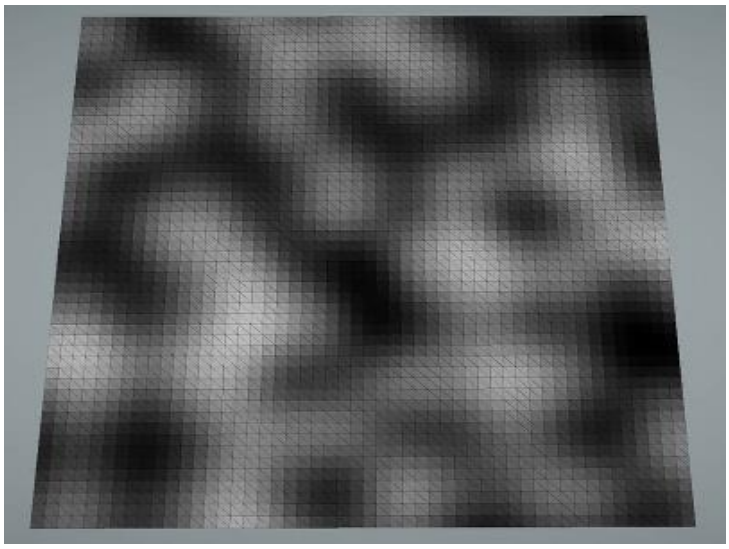


Рис. 3.3 – меш-сітка з накладеною текстурою шуму Перліна

Силу накладання шуму, розмір та положення можна змінювати у налаштуваннях (див. Рис. 3.4)

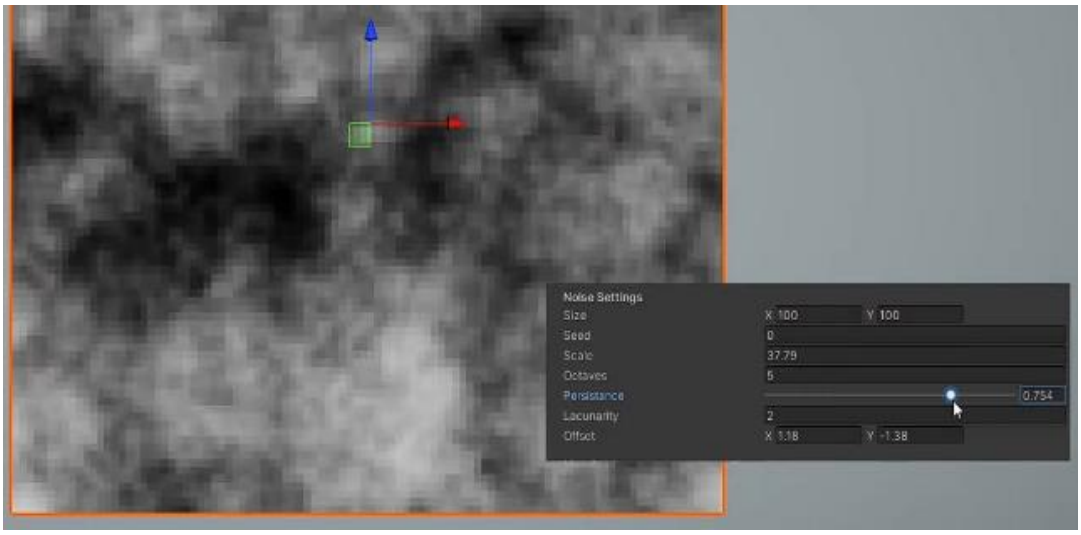


Рис. 3.4 – Регулювання характеристик шуму всередині рушія

А також, утворену 3D реалізацію алгоритму можна використати як декорацію ландшафту. За потреби, для випадкового розміщення кімнат можна застосувати інший алгоритм, а функцію шуму залишити виключно в якості декорації.

У визначені точки розміщуються самі кімнати (див Рис. 3.5), в даному випадку вони будуть круглими, таким чином буде простіше будувати з’єднання між ними, крім того, з художньою метою стін у кімнат не буде, гравець зможе впасти з платформи і програти, це буде додатковим елементом складності.

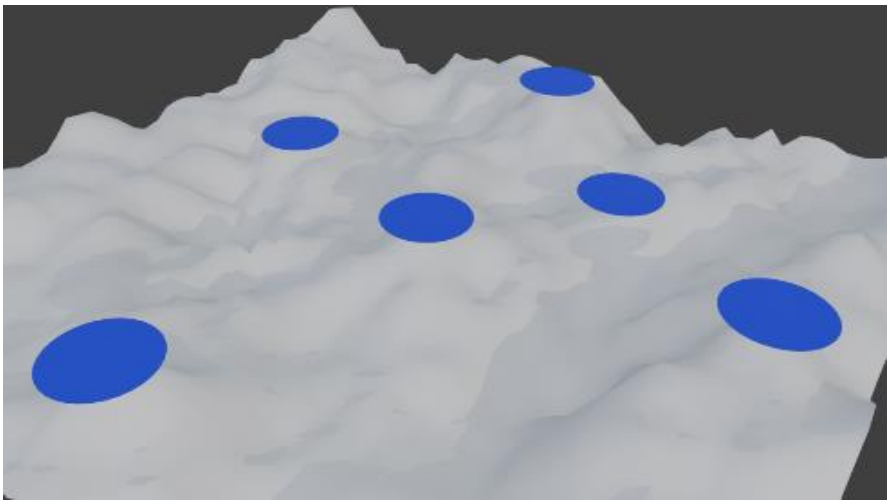


Рис. 3.5 – Розміщення кімнат на рівні

Побудова коридорів

Побудова коридорів відбувається з використанням графу на основі тріангуляції Делоні. Такий алгоритм будує зв’язки між всіма вузлами, що дозволяє в подальшому, використовуючи алгоритм кістякового дерева знайти оптимальні варіанти. Також можна додати додаткові шляхи для утворення більш цікавої структури рівня. Після додавання ігрових об’єктів, що відображають коридори візуально, матимемо наступний результат (див. Рис. 3.6).

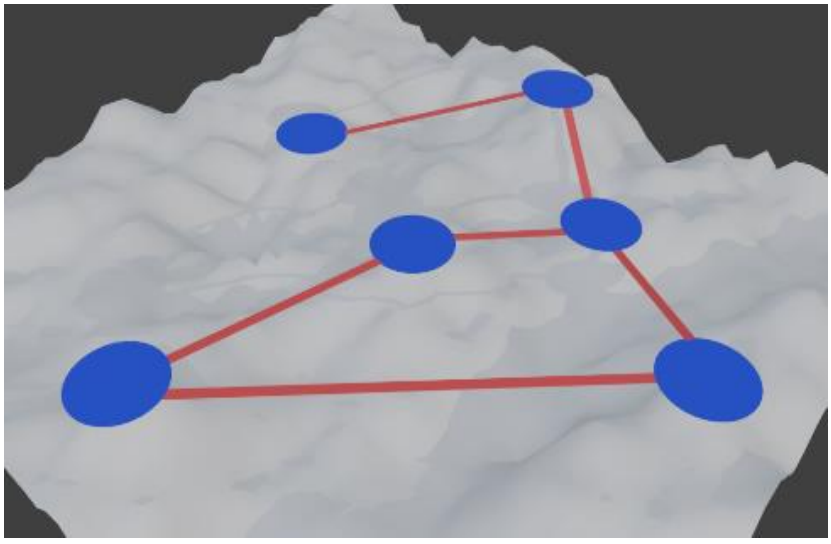


Рис. 3.6 – Кімнати з’єднані коридорами

Концептуально рівень представляється у вигляді певного ландшафту, на вершинах якого є павучі гнізда (кімнати), що з’єднані павутиною (коридори). Гравцеві потрібно переміщатись між кімнатами рівня по тоненьким смугам, на які можна додатково додати певні перешкоди. Оскільки передбачена

можливість падіння з платформ рівня, є можливість урізноманітнити геймплей через генерацію більшої кількості коридорів між кімнатами, аби у гравця була можливість перестрибувати між ними. На Рис. 3.7 зображений приклад накладання коридорів.

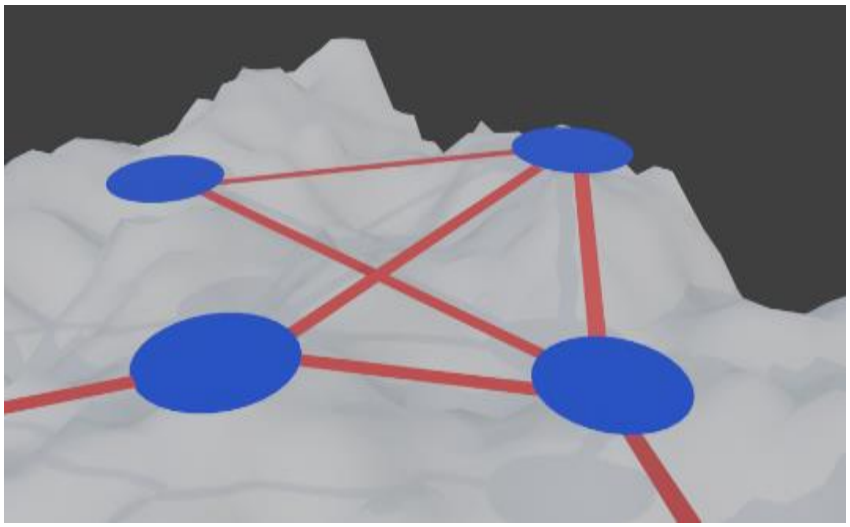


Рис. 3.7 – Накладання коридорів

В подальшому можна накладати стилізовані текстури на ігрові елементи рівня, наприклад зробити коридори у формі реалістичного павутиння, з ефектами прозорості та відповідними текстурами. Але ми зосередимось на процедурних аспектах.

Процедурна анімація

Наступний процедурний метод, який буде використано в проєкті, це процедурна анімація, вона використовує алгоритми та математичні моделі для автоматичного створення рухів, що дозволяє зменшити потребу в ручному створенні анімацій для кожного кадру. Це робить процес ефективнішим і дозволяє зекономити час і ресурси, оскільки об'єм роботи значно скорочується.

Процедурна анімація є надзвичайно гнучкою, оскільки може адаптуватися до змін в реальному часі — наприклад, залежно від взаємодії персонажа з навколишнім середовищем або фізичних умов, таких як гравітація, тертя чи зіткнення. Ця адаптивність дозволяє створювати більш природні і непередбачувані рухи, що особливо корисно в іграх та симуляціях.

Іншою важливою перевагою процедурної анімації є її здатність до масштабування та інтеграції з фізичними симуляціями. Вона дозволяє

анімувати великі сцени або безліч об'єктів одночасно, без необхідності вручну задавати кожну деталь. Це значно знижує витрати часу та вартості розробки, що особливо актуально для великих проєктів з динамічними світом чи великою кількістю об'єктів. Процедурні методи також дозволяють забезпечити високу інтерактивність, де анімація змінюється залежно від дій користувача, що створює більш захоплюючий і персоналізований досвід.

В даному випадку, оскільки ігровий персонаж, очевидно, певний павучок, процедурно буде виконуватись анімація ніжок. Процедурна анімація передбачає закладання правил в амплітуду анімації, і відповідно від середовища ці правила будуть виконуватись.

Наприклад, якщо виконується рух, відповідні анімовані об’єкти реагуватимуть на зміну координат та програватимуть анімацію в темп швидкості зміщення. Якщо ландшафт передбачає зміщення лінії горизонту у вигляді певних перешкод, анімація відповідно реагуватиме на зміну по осі Z.

Побудова процедурної анімації ніг відбувається за таким принципом:

1. Використання зворотної кінематики для керування ногою

2. Фіксування нижнього елементу ноги відповідно поверхні, в той час як тіло рухається вільно паралельно землі в напрямку руху.

3. Створюється своєрідний сенсор, у вигляді прив’язаного до тіла моделі об’єкта. (наприклад сфера, що рухається залежно від тіла але на рівні поверхні) Сенсор реагує на колізію, при цьому він відв’язаний від об’єкта тіла по осі Z. Таким чином сенсор розпізнає перешкоди і «плаває» по ним по вертикалі.

4. Наступний крок це прив’язка низу ноги до сенсору згідно ліміту на відстань. Таким чином, якщо сенсор, слідуючи за тілом віддаляється від зафіксованої на площині нижньої точки ноги, остання тягнеться до поточної позиції сенсору утворюючи крок. Тобто тепер, якщо тіло рухається, сенсор і «смикає» в певний момент ногу. Маємо процедурну анімацію крокування.

5. Далі необхідно утворити ліміти вертикального положення тіла, які визначатимуться через середню відстань до сенсору, схожим принципом, як від нього залежать ноги.

6. Для природності анімації, варто додати правило почергових рухів

декількома ногами, залежно положення протилежної ноги. Також можна додати нахилення тіла залежно від положення ніг.

Процедурна анімація виглядає більш природно та плавно на відміну від звичайної, вона не залежить від попередньо записаних кадрів чи рухів, а реалізується в реальному часі, пристосовуючись до подій у сцені згідно заданим правилам процедури.

З очевидних причин продемонструвати роботу анімації у вигляді скриншоту не є об’єктивно, тож обмежимось простим поясненням алгоритму.

Приблизний вигляд який матиме ігровий персонаж зображено на Рис. 3.8.

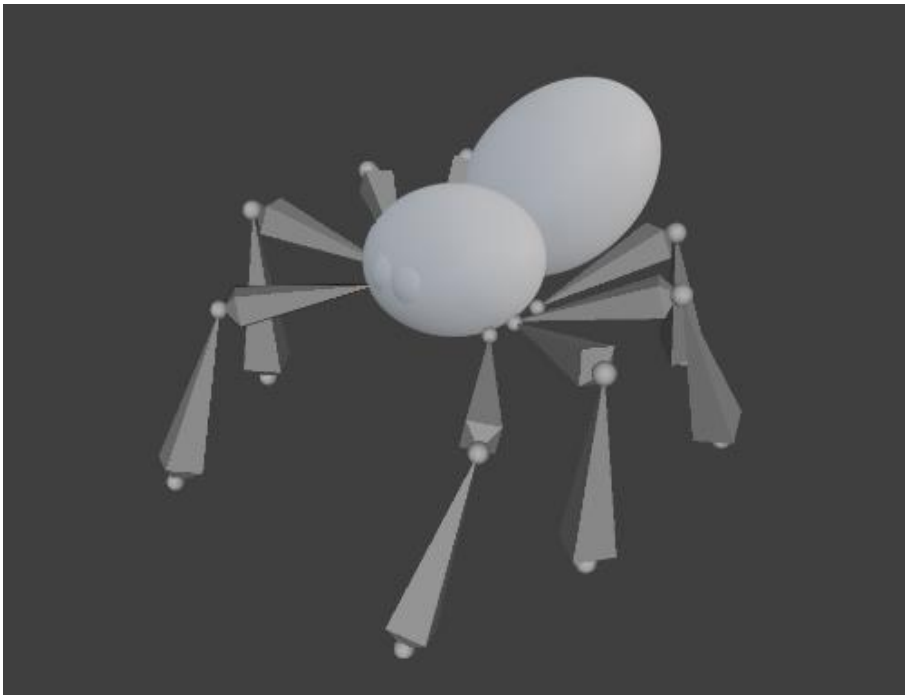


Рис. 3.8 – Приблизний вигляд (концепт) ігрового персонажа

Подальший розвиток проєкту передбачає більш глибоке налаштування ігрових механік, додавання більшої кількості текстур та уваги до дизайну, збільшення кількості ітерацій і даних для алгоритмів генерації, інтеграція систем нагород, з метою зробити ігровий процес більш цікавим залежно від потреб кінцевого споживача, за для успішної комерціалізації та популяризації проєкту.

Висновки до розділу 3.

У даному розділі було виконано огляд програмних засобів, які необхідні для виконання проєкту. Ігровий рушій Unity чудово підходить як для реалізації невеликих проєктів чи демоверсій, так само і для високобюджетних розробок. Відкритий доступ до ліцензії та можливість розробок власних інтеграцій, робить його зручним та потужним інструментом для розробки різного роду застосунків чи цифрового контенту.

Unity також є кросплатформним рушієм, тому розроблені програми можуть одночасно підтримуватись на кількох цільових платформах. Unity також має пряму інтеграцію з VisualStudio, рушій дозволяє використання C# для додавання кастомних скриптів, а також підтримує безліч спеціалізованих бібліотек, які користувачі можуть використовувати або розміщувати у відкритому доступі власні. Для цього існує окрема платформа Unity Asset Store, де розробники можуть розміщувати власні надбудови у відкритому доступі або за певну плату.

Також було виконано процес розробки програми відповідно до поставленої задачі. В програмі продемонстровано застосування основних методів процедурної генерації у різних напрямах, зокрема

- шум Перліна для побудови ландшафту та розрахунку випадкових позицій для розміщення кімнат рівня;
- алгоритму на основі графу, для обчислення графу зв’язків між кімнатами;
- впорядкування графа через алгоритми пошуку шляху, для визначення оптимальної побудови коридорів між кімнатами;
- створення процедурної анімації

У підсумку розроблена програма демонструє різнобічність застосувань та різноманітність методів процедурної генерації.

ВИСНОВКИ

Кваліфікаційна робота присвячена процедурній генерації, зокрема її застосуванню для генерації унікального контенту в розважальних сферах. Досліджено основні причини виникнення потреб у застосуванні процедурної генерації та шляхи популяризації використання алгоритмів цього типу.

В ході роботи було проведено дослідження основних методів процедурної генерації і напрямів їх застосування, зокрема в жанрі roguelike. Було досліджено походження жанру, підстави його виникнення та шляхи популяризації, а також вплив на подальший розвиток індустрії і використання методів генерації. Також було досліджено безпосередній розвиток самих алгоритмів у різних сферах використання, зміну типу генерованого контенту, ступенів його унікальності та характеру застосування.

Було досліджено основні принципи виконання алгоритмів шуму Перліна, його розвиток у алгоритм Simplex шуму, їх використання в генерації текстур та ландшафтів, можливості потенційного застосування в інших сферах. Дослідження торкнулось алгоритмів процедурної генерації ігрових рівнів, серед них алгоритми на основі генерації кімнат та коридорів, клітинних автоматів, поділу простору, «дріблення стін», алгоритми генерації лабіринтів Пріма та Крускела, алгоритми на основі графів. Також було досліджено роботу сучасного алгоритму заснованого на ідеях суперпозиції у сфері квантової механіки, а саме, алгоритму Wave Function Collapse та різних його варіацій виконання для процедурної генерації в різних вимірах.

Розроблена програма виконує одразу кілька різних методів процедурної генерації і відповідає поставленим задачам. Зокрема, програма об’єднує:

- Алгоритм процедурної генерації ландшафту на базі шуму Перліна
- Алгоритми побудови та визначення графів для створення зв’язків в структурі рівня
- Алгоритми побудови процедурної анімації для автоматизації адаптації анімацій під ігровий процес

Робота виконана в жанрі roguelike та має високу варіативність в плані процедурної генерації контенту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Harrison Ferrone (2021). Learning C# by Developing Games with Unity 2020 (5th edition).

2. Джейсон Шраєр. Кров, піт і пікселі. По той бік створення відеоігор. Book Chef, 2020. 336 с.

3. Paris Buttfield-Addison, Jonathon Manning, Tim Nugent. (2019) Unity Game Development Cookbook: Essentials for Every Game (1st edition).

4. Jesse Shell (2018) The Art of Game Design: A Book of Lenses (3rd edition).

5. Noor Shaker, Julian Togelius, Mark J. Nelson (2016) Procedural Content Generation, Springer, 2016.

6. Julian Togelius (2019) Playing Smart: On Games, Intelligence, and Artificial Intelligence. MIT Press, 2019.

7. Compton, K. & Mateas, M. (2016) Procedural Generation in Game Design. AK Peters/CRC Press, 2016.

8. Whitehead J. (2018) Generating Games and Game Content: A Procedural Approach. Morgan & Claypool Publishers, 2018.

9. Ryan Watkins (2016) Procedural Content Generation for Unity Game Development, 260 pages

10. McGuire, M., & Jenkins, O. C. (2017) Creating Games with AI: Using Procedural Generation and AI Techniques. CRC Press, 2017.

11. Adam Newgas. (2021) Tessera: A Practical System for Extended WaveFunctionCollapse. In The 16th International Conference on the Foundations of Digital Games (FDG) 2021 (FDG'21), August 3–6, 2021, Montreal, QC, Canada. ACM, New York, NY, USA, 7 pages

12. Adam Newgas. 2024. What Is Procedural Generation [Електронний ресурс]/ boristhebrave – Посилання на ресурс: https://www.boristhebrave.com/2024/05/25/what-is-procedural-generation/

13. Adam Newgas. 2021. Lock and Key Dungeons [Електронний ресурс]/ boristhebrave – Посилання на ресурс: https://www.boristhebrave.com/2021/02/27/lock-and-key-dungeons/

14. Adam Newgas. 2021. Graph Rewriting for Procedural Level Generation

[Електронний ресурс]/ boristhebrave – Посилання на ресурс: https://www.boristhebrave.com/2021/04/02/graph-rewriting/

15. Adam Newgas. 2020. Wave Function Collapse Explained [Електронний ресурс]/ boristhebrave – Посилання на ресурс: https://www.boristhebrave.com/2020/04/13/wave-function-collapse-explained/#back-to-wfc

16. Adam Newgas. 2020. Wave Function Collapse tips and tricks [Електронний ресурс]/ boristhebrave – Посилання на ресурс: https://www.boristhebrave.com/2020/02/08/wave-function-collapse-tips-and-tricks/

17. Jason McCollum. 2022. An introduction to graph rewriting for procedural content generation [Електронний ресурс]/ shaggydev – Посилання на ресурс: https://shaggydev.com/2022/11/20/graph-rewriting/

18. Добірка рушіїв для розробки ігор [Електронний ресурс]/DOU: 2021 – Посилання на ресурс: https://gamedev.dou.ua/blogs/game-engines-collection/?from=similar_posts

19. Кастомні рандомогенератори та їх реалізація в Unity ігор [Електронний ресурс]/DOU: 2024 – Посилання на ресурс: https://gamedev.dou.ua/forums/topic/47506/?from=similar_topics

20. Maxim Gumin. 2016. WaveFunctionCollapse. [Електронний ресурс]/GitHub repository – Посилання на ресурс: https://github.com/mxgmn/WaveFunctionCollapse

21. Oskar Stålberg. Wave [Електронний ресурс]/ oskarstalberg blog – Посилання на ресурс: https://oskarstalberg.com/game/wave/wave.html

22. Martin Donald. Wave Function Collapse – Initiative Demo [Електронний ресурс]/ bolddunkley – Посилання на ресурс: https://bolddunkley.itch.io/wfc-mixed

23. Robert Heaton. 2018. The Wavefunction Collapse Algorithm explained very clearly [Електронний ресурс]/ RobertHeaton blog – Посилання на ресурс: https://robertheaton.com/2018/12/17/wavefunction-collapse-algorithm/

24. Generating World Maps for Unexplored 2 [Електронний ресурс]/gamedeveloper: 2019 – Посилання на ресурс:

https://www.gamedeveloper.com/design/generating-world-maps-for-unexplored-2

25. Procedural Map Generation [Електронний ресурс]/gridsagegames: 2014 – Посилання на ресурс: https://www.gridsagegames.com/blog/2014/06/procedural-map-generation/

26. Kate Compton. 2016. So you want to build a generator… [Електронний ресурс]/ galaxykate0 – Посилання на ресурс: https://galaxykate0.tumblr.com/post/139774965871/so-you-want-to-build-a-generator

27. Procedural generation [Електронний ресурс]/Wikipedia: 2024 – Посилання на ресурс: https://en.wikipedia.org/wiki/Procedural_generation#

28. Perlin noise [Електронний ресурс]/Wikipedia: 2024 – Посилання на ресурс: https://en.wikipedia.org/wiki/Perlin_noise#

29. Simplex noise [Електронний ресурс]/Wikipedia: 2024 – Посилання на ресурс: https://en.wikipedia.org/wiki/Simplex_noise

30. Maze generation algorithm [Електронний ресурс]/Wikipedia: 2024 – Посилання на ресурс: https://en.wikipedia.org/wiki/Maze_generation_algorithm

31. Procedural generation: Creating infinite algorithmic realities [Електронний ресурс]/ autodesk: 2023 – Посилання на ресурс: https://www.autodesk.com/solutions/procedural-generation

32. Bifrost for Autodesk Maya Helps Pixomondo Streamline VFX Creation [Електронний ресурс]/ autodesk: 2023 – Посилання на ресурс: https://blogs.autodesk.com/media-and-entertainment/2023/10/15/bifrost-for-autodesk-maya-helps-pixomondo-streamline-vfx-creation/

33. Unity Documentation [Електронний ресурс]/Unity: 2024 – Посилання на ресурс: https://docs.unity3d.com/Manual/index.html

34. Unity Products [Електронний ресурс]/Unity: 2024 – Посилання на ресурс: https://unity.com/products

35. Unity Learn [Електронний ресурс]/Unity: 2024 – Посилання на ресурс: https://learn.unity.com/