Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Модель процедурної генерації
програмними засобами для спрощення
розробки додатків
(тема)

Виконав:

студент II курсу, групи СПм-21-1
Кісь О.В.
(прізвище, ініціали)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

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

Освітня програма
Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: Ляшенко О.С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

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

Освітня програма Комп’ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Кісю Олександру Вікторовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Модель процедурної генерації програмними засобами для спрощення розробки додатків

затверджена наказом по університету від “ 07 ” 11 2022 р. № 1454Ст

2. Термін подання студентом роботи до екзаменаційної комісії 13 грудня 2022р.

3. Вхідні дані до роботи мова C#, технологія .NET, Unity

4. Перелік питань, що потрібно опрацювати у роботі

1) Аналіз проблеми та огляд існуючих рішень
2) Формування алгоритму процедурної генерації плиткового рівня
3) Програмна розробка алгоритмів генерації
4) Тестування програмних модулів
5) Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Слайд-презентація – 15 слайдів

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

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Видача теми проекту, узгодження і затвердження теми	05.10.2022-07.11.2022	
2	Аналіз існуючих рішень,алгоритмів та постановка задачі	08.11.2022-17.11.2022	
3	Розробка програмної частини	18.11.2022-26.11.2022	
4	Оформлення пояснювальної записки	29.11.2022-06.12.2022	
5	Перевірка виконаного проекту керівником	07.12.2022-09.12.2022	
6	Захист роботи	10.12.2022-22.12.2022	

Дата видачі завдання 07.11.2022

Студент _____
(підпис)

Керівник роботи _____ доцент, к.т.н. Ляшенко О.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 90 сторінок, 5 частин, 2 додатки, 43 рисунків, 1 таблиця, 24 джерел.

ГРА, СИСТЕМИ, ГРАФ, ГЕНЕРАЦІЯ, АЛГОРИТМ, ГРАВЕЦЬ, ЛАБІРИНТ, МЕНЮ, МЕНЮ, C#, UNITY ,СКАРБИ, КАРТА.

Об’єктом дослідження даної кваліфікаційної роботи є алгоритми процедурної генерації та середовища Unity

Метою роботи є створенння фунціоналу для комбіновоної процедурної генерації різноманітних підземель, ландшафту та отримання фундаментальних навичок роботи з Unity, розуміння концепції генерування випадкових елементів, освоєння основних принципів документації програм.

В результаті було розроблено додаток, який описується в даній пояснювальній записці.

ABSTRACT

Bachelor’s thesis 90 pages, 43 figures, 2 appendices, 1 table, 24 sources.

GAME, SYSTEMS, GRAPH, GENERATION, ALGORITHM, PLAYER, LABYRINTH, MENU, MENU, C #, UNITY, TREASURES, MAP.

The object of research of this course work are algorithms of procedural generation and Unity environment

The purpose of the work is to create a functional for the combined procedural generation of various dungeons, landscape and gain basic skills of working with Unity, understanding the concept of generating random elements, mastering the basic principles of program documentation.

As a result, an appendix was developed, which is described in this explanatory note.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕН, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Геймплей – ігровий процес;

Ігровий движок – набір програмного забезпечення для спрощення створення відеогри як програмного продукту;

Кросплатформність – властивість програмного забезпечення працювати більш ніж на одній програмній (в тому числі операційній системі) або апаратній платформі;

Патерн – шаблон проектування архітектури додатку;

Префаб – ігровий об'єкт призначений для повторного та багатократного використання у ігрових сценах;

Рендер – процес відрисування математичної моделі об'єкту як зображення;

Скрипт – файл з кодом, що описує поведінку об'єкту;

Скриптинг – написання сценаріїв поведінки об'єктів на мові програмування;

Фреймворк – інфраструктура програмних рішень, що полегшує розробку складних систем, що має ряд правил та обмежень, що задають правила створення структури проєкту та написання коду;

GameObject – ігровий об'єкт;

.NET – це платформа від Microsoft, яка дозволяє створювати програмні додатки;

ВСТУП

Процедурна генерація - це метод створення вмісту алгоритмічно, а не вручну. У відеоіграх він часто використовується для підвищення можливості відтворення гри. Класичним прикладом є гра Diablo, яка є відеоіграми про сканування підземель, натхненною настільною грою Dungeons & Dragons. Він містить генеровані процедурними рівнями підземель, скарби та зустрічі з монстрами, а також все, що призводить до неповторного досвіду при кожному проходженні.

Процедурна генерація часто використовується створення ігрових рівнів. Одним із поширених підходів до цієї проблеми є використання бінарного поділу простору. Алгоритм починається з прямокутної області та рекурсивно ділить її, доки не буде достатньо підобластей. Потім виділяють кілька підрайонів, що становлять кімнати, і з'єднують їх коридорами. Інший можливий підхід – так звані агентські підземелля. Алгоритм починається з області, повністю заповненої осередками стіни, та агенти з'являються у зазначених місцях.

Проблема з цими алгоритмами полягає в тому, що геймдизайнери часто втрачають контроль над ходом ігрового процесу, і макети, що виходять, здаються занадто випадковими і позбавленими загальної структури. Ці ігри засновані на сюжеті з такими концепціями, як вирішення головоломок та дослідження, які становлять більшу частину ігрового процесу. Вони прагнуть вирішити цю проблему, використовуючи генеративні граматики для створення місій і ігрових просторів. Їх метод приймає набір форм кімнати як вхідні дані та створює макет, який відповідає певній структурі зв'язаного графа або певній логіці поколінь.

Метою даної кваліфікаційної роботи є створення алгоритму з додатком, який дозволить контролювати наповнення контенту при генерації нових підземель roguelike стилю та доступу до них через згенерований ландшафт.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Найбільш важливим етапом проектування додатку будь-якого типу є аналіз предметної області, що закінчується побудовою структурної (концептуальної схеми).

Для цього етапу є характерним аналіз потреб користувачів, вибір основних інформаційних об'єктів та суб’єктів, їх характеристики. Тоді, на основі виконаного аналізу створюється структура предметної області, яка не залежить ні від програмного, ні від апаратного середовища, в якому буде реалізовуватися додаток. Тобто, складається певний набір вимог і схем, що можуть бути реалізовані багатьма шляхами.

Створення ігрових рівнів є багатогранною задачею, актуальною для більшості випущених ігор. Найчастіше створенням рівнів займається окрема людина - дизайнер рівнів. При розробці великих проектів створенням рівнів може займатися і команда людей.

Дизайн рівнів як процес включає безліч питань. Одним із найважливіших є обсяг роботи, необхідний для створення всіх рівнів гри. Звичайно, масштаб ігор і природа віртуальних просторів, що реалізуються ними, зазвичай сильно різняться, але часто чим більше в грі рівнів і чим вони більш опрацьовані і деталізовані, тим більш конкурентоспроможною стає сама гра. З цього можна зробити висновок про актуальність до прагнення мінімізувати витрати на дизайн рівнів при одночасному досягненні максимальної кількості та якості.

1.2 Аналіз існуючих розробок

Жанр Roguelike - жанр комп'ютерних ігор, є піджанром рольових

покрокових ігор. Даний жанр має деякі відмінні особливості, серед яких можна виділити наступні:

випадкова генерація ігрового оточення (локація, вороги, бонуси та інше), яка є унікальною для кожного нового проходження;

пошаговость дій, кожна команда повинна дорівнювати одному ходу або одній дії;

необратимість дій, будь-яка дія може призвести до фатальної помилку, аж до остаточної загибелі персонажа і неможливості згодом продовжити гру;

повна доступність всіх ігрових дій з самого початку гри, тобто у гравця спочатку є весь доступний функціонал і він може ним користуватися;

самостійне дослідження навколишнього світу, гравець повинен сам досліджувати і розібратися в роботі всіх ігрових механік;

ігрок має повну свободу дій, в грі не передбачено лінійного проходження.

NetHack - одна з найбільш відомих roguelike ігор. Доступ до гри відбувався в 1987 році. «Net» - це позначка, яка має на увазі, що даний продукт не розрахований на багато користувачів і має лише одиничний режим гри, як може здатися на перший погляд, то революційності у розробці нових ігор того часу не має, але по-перше велике значення зайняв при розробці відкритий діалог команди розробників з майбутніми гравцями, а по-друге, активне обговорення проходження продукту та знайдені особливості з секретами у мережі інтернету . «Hack» в назві відсилає до одного з відгалужень жанру ролі-вих ігор - hackandslash (знищення великої кількості ворогів в ближньому бою) - що дуже точно описує ігровий процес.

NetHack вважається однією з найстаріших ігор, яка до сих пір підтримує-ся і розвивається розробниками (останнє оновлення гри вийшло навесні 2018 року). Завдяки відкритому коду дана гра була перенесена на величезну кількість платформ (Linux, FreeBSD, MacOS, iOS, Androidі інші). Гра виконана з використанням псевдографічні графіки проте деякі порти мають тайловую графіком.

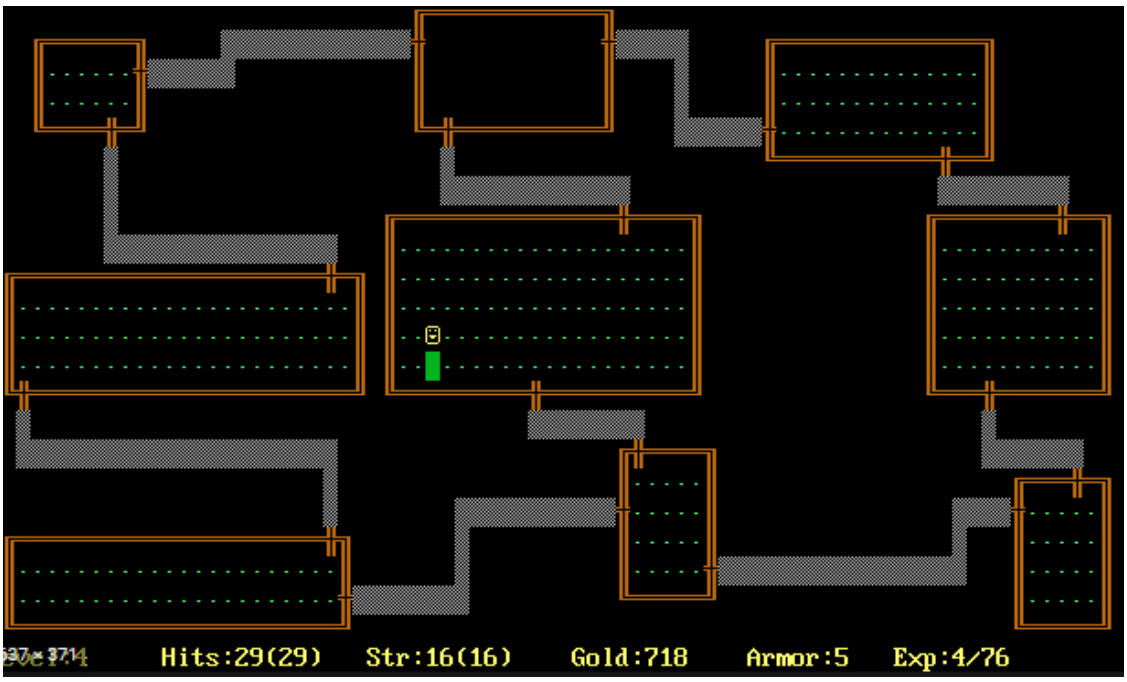

Рисунок 1.1 – Гра «NetHack»

Був проведений аналіз декількох ігор даного жанру різних років випуску.

The Binding of Isaac – особливістю данної гри, це 2D-RPG з видом зверху,

де гравець керує одним з шести доступних героїв, але в основі сюжету постає Ісаак. Випущена гра будела у 2011 та отримала велику кількість нагород, дійсно гра зайняла почесне місце серед кращих та відомих rougilike ігор. Механіка і зовнішнє оформлення гри нагадує ранні гри серії The Legend of Zelda, але з додаванням елементу випадковості, незворотності смерті і процедурно генеруються рівнів - механік, характерних для жанру roguelike. На кожному поверсі підземелля гравець змушений битися з монстрами в кожній кімнаті, щоб просуватися далі. По дорозі він може збирати гроші, щоб купити спорядження в магазині, ключі, щоб відкривати скарбниці, нова зброя і різні посилення, щоб збільшити свою перевагу перед ворогами. На кожному поверсі підземелля присутній бос, якого потрібно вбити для отримання можливості переміститися до наступного рівня.

Рисунок 1.2 – Гра «The Binding of Isaac»

Яскравим прикладом буде «World of Warcraft » , дана гра вже давно увійшла в історію, як одна з найпопулярніших, революційних та орієнтовну на на багато користувачів ролева онлайн грою, розроблена компанією Blizzard Entertainment. Основна історія та дії в World of Warcraft відбувається в однойменному всесвіту Warcraft, який детально був описаний у попередніх іграх серії та з кожним оновленням лише розширяється. Основна ідея гри - керуванням буквально життям свого героя у реальному часі в онлайн світі, де можливе проходження підземель, підняття рівнів, отримання предметів, бої з босами та боротьба між гравцями. Гра була випущена 23 листопада 2004 року, а у листопаді 2020 року отримала доповненя «Shadowlands», де розробники у новій варіації продемонстрували 10 варіацій генераторів підземель , кожний зі своєю особливістю. А вже 28 листопада 2022 року відбулося оновлення «Dragonflight», в якому використовується поодиночна генерація об’єктів у відкритому світі та особливому контенті – підземелля. Недивлячись на те ,що грі вже понад 16 років, ця гра має величесну аудиторію посьогодні, а поява процедурної генерації в ній підтверджує

актуальність та потрібність в наш час.

Рисунок 1.3 – Гра «World of Warcraft: Shadowlands»

Рисунок 1.4 – Гра «World of Warcraft: Shadowlands»

Важливо розібрати і ігри в основі яких використовується процедурна

генерація ландшафту. Одним з яскравих представників розробленим у 2016 році є «No Man’s Sky». В основі грі постає вивчення космосу, планет, тварин, отримання ресурсів, де саме все генерується випадковим чином. Кожна ж частина космосу унікальна, а можливість до гри с іншими людьми надали грі величезної популярності. Легка доступність до коду надала можливість створення модифікацій, які підтримуються навість на даний момент.

Рисунок 1.5 – Гра «No Man’s Sky»

Дійсно легендарною стала гра «Minecraft» 2009 року. В розпорядженні гравця постає процедурно генерований тривимірний світ, що повністю складається з кубів. Майже все можна перебудувати, а світ чимось схожий на конктруткори LEGO, тому величезна кількість модельєрів та будівників отримали простір до реалізації своїї фантазій. Основна перевега даного продукту, що перед гравцем не має конкретних цілей, таким чином кожен може робити будьщо згідно до своїх потреб, як наприклад дослідження світу, добувати корисні копалини, боротися з противниками і багато іншого.

Рисунок 1.6 – Гра «Minecraft»

Одним з останніх дійсно ігрових проривів для людей зробила гра «Valheim» 2021 року, під час активного періоду пандемії люди з усіх куточків світу підняли рейтинг продаж до лідерства та дуже довгий час не опускали рівень гри. Основа гри це виживання з елементами RPG і «Пісочниці». Головним героєм у історії постає вікінг у всесвіті Вальхейм, згідно до історії він відноситься до 10 світу з скандинавської міфології. Ігровий процес не лише надає гравцю можливість будувати побут, але й боротися з численними чудовиськами. Також в грі можна подорожувати на кораблях-драккарах, будувати застави, замки, шукати їжу, добувати речі. Світ тут випадково-генерований, а ще можливо грати з іншими людьми.

Рисунок 1.7 – Гра «Valheim»

З цих прикладів можна підмітити,що інтерес гравців до ігор з часом лише зростає і при створенні ігор важливу частину відіграє графіка та динамічний геймплей, використаємо ж це при створенні своєї гри.

Проаналізувавши представлені приклади ігор, можна зробити висновок, що це є актуальним в наш час. Завдяки актуальності даного дослідження було вирішено зробити додаток в основі маючий процедурну генерацію локації. Програма може використовуватись ,як розважальний матеріал, оскільки має динамічну структуру і може задовольнити більшість користувачів.

1.3 Аналіз існуючих алгоритмів

1.3.1 Алгоритм на основі BSP-дерева

Двійкове розбиття простору (BSP - binary space partition) - це метод,

рекурсивного поділу простору на два опуклих набори. Вперше був сформульований в 1969 році в контексті тривімірної комп'ютерної графіки. Розробка алгоритму насамперед замислювалася задля сортування полігонів в сцені, породжуючи просторову інформацію про об'єкти у вимірі. У ті часи не існувало апаратної підтримки буфера Грубін (z-buffer), а існуюча програмна реалізація буфера була занадто повільної. Проте метод BSP залишається актуальним навіть після створення апаратного z-buffer-а. Опріч сортування, метод широко застосовується і в інших галузях комп'ютерних обчислень, таких як перевірка на зіткнення, в деяких алгоритмах комп'ютерних мереж або в методі випромінювання.

Розглянемо послідовність виконання двійкового розбиття простору. Спочатку до першого вузла(кореня) дерева привласнюється прямокутний рівень заповнений стінами, чи просто пустим простором. Надалі область рівня вузла поділяється на дві підобласті котрі діляться на два дочірніх вузла. Напрямок ділення, а саме вертикальне чи горизонтальне, обирається випадковим чином, після чого надається перевага відповідній координаті ділення. В результаті маємо дві області меншого розміру. Після чого ця послідовність розподілу, як було вказано, рекурсивно повторюється, включаючи отримані підобласті. Процес триває й надалі, поки не задовольнить обрані нами вимоги. Для прикладу, поки розмір кожної розглянутої області не стане занадто маленьким хоча б по одній з осей. Розглянемо детальніше: задається, що ділянка може ділитися тільки тоді, коли і її ширина, і її висота не менше 20 клітин, і при цьому координата ділення не може стояти ближче, ніж на 10 клітин до області кордону. Це гарантує, що всі кінцеві ділянки будуть мати розмір не менше ніж 6 клітин по горизонталі та 6 клітин по вертикалі.

Також можливо вказати додаткову умову виходу залежну від кількості ділянок (скажімо, завершити процес розбиття, якщо областей стало більше 10). Якщо початкові розміри рівня значно більше, ніж мінімальна площа однієї ділянки, помножена на максимальну кількість цих ділянок, то

вірогідно, що саме обмеження на кількість кімнат буде слушною та остаточною вимогою для завершення алгоритму, а не досягнення мінімальних розмірів кожної з областей. В такому випадку більш доречно використовувати саме ітеративний варіант алгоритму, а не рекурсивний, і вже після кожного ділення ділянки сортувати поточний список областей зі зменшенням їх площи. Тоді розміри кімнат будуть розподілені відносно рівномірно між собою, що являє собою бажаний результат.

Цей процес поділу областей на підобласті породжує та вибудовує деревоподібну структуру даних, завдяки якому алгоритм і носить свою назву. У кожній вершині(вузлі) дерева зберігається інформація про дані розподілу об'єкта, як приклад, у вузлі зберігаються координати лівого верхнього кута та ширина і висота діленої ділянки, а також, в будь-якому випадку, посилання на двох нащадків - якщо підобласті є, або ж порожні посилання - якщо даний вузол кінцевий. Перша вершина(корінь) дерева зберігає відомості про ділянку всього рівня чи об'єкта цілком.

Рисунок 1.8 – Результат створення кімнат

Створюється граф Тріангуляція Делоне. Задля розуміння поданої далі інформації розглянемо поняття більш детально.

Тріангуляція, в контексті комп'ютерної тривімірної графіки, це процес коли кожний або обраний полігон сітки моделі, котрий складається з N вершин, ділять та об’єднують ребрами(зв’язок між вершинами), що не перетинаються, в нові полігони з трьома(і тільки трьома) вершинами.

Тріангуляція Делоне це процесс в якому всі трикутники(полігони з трьома вершинами) сітки прагнуть до правильної форми. З чого випливає, що для множини точок на площині - це така тріангуляція, що жодна вершина множини не буде знаходитися всередині описаних довкола трикутників кіл в множині. Тобто, якщо провести коло через три довільні вершини, то більше в колі не буде інших вершин. Це називається круговий критерій. Через це тріангуляція Делоне дозволяє зменшити кількість малих кутів, що надає більшу оптимізацію для важких обчислень. Цей спосіб тріангуляції було

названо в честь свого винахідника – математика Бориса Миколайовича Делоне, в 1934 році.

На основі визначення поняття Делоне ми отримаємо, що коло описане навколо трикутника та сформоване трьома вершинами з отриманої множини цих точок буде називатися пустим, якщо воно не містить вершин трикутника інших ніж ті три, що вже вказані. Інші точки можуть допускатися тільки якщо вони на периметрі кола, але не на його площині.

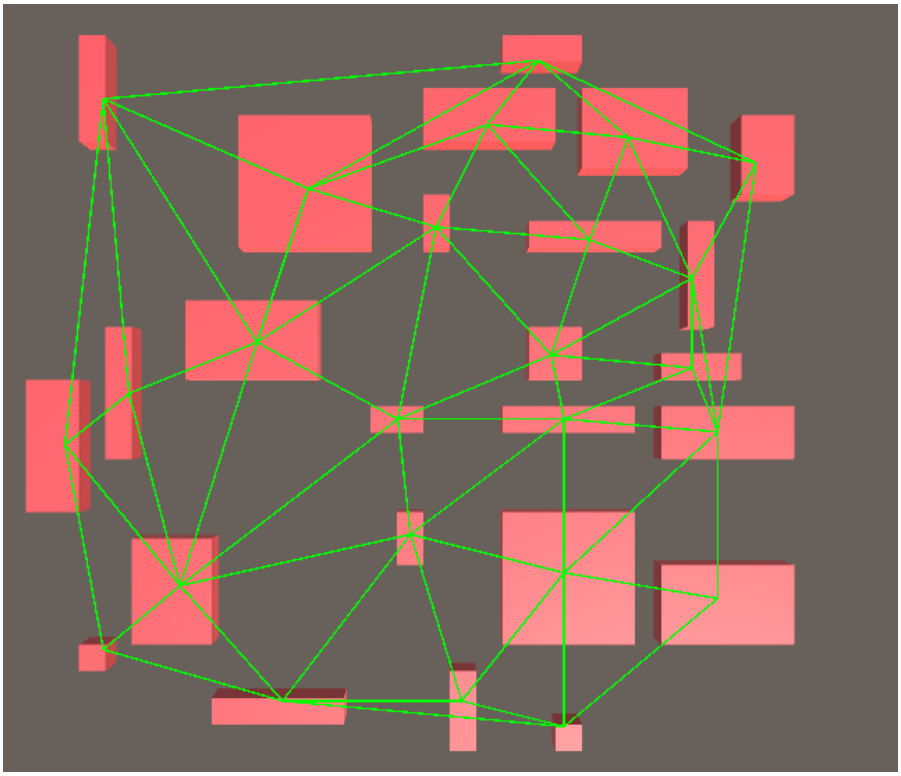

Рисунок 1.9 – Граф Тріангуляція Делоне

Маючи результат тріангуляції у вигляді полігональної сітки, ми можемо створити на його основі мінімальне кістякове дерево. Потрібно зауважити, що кістяк – це деревоподібний підграф, що має максимальну кількість окремих ребер без циклів і при цьому сума всіх реберних довжин, включних у кістяк, буде мінімальною. Також треба розуміти що кістяком називається зв’язний граф, а от наявність декількох незв’язаних ліній у

вихідному графі називається кістяковий ліс.

Отже, підсумовуючи, отримаємо, що мінімальне кістякове дерево у неорієнтованому графі, котрий є зваженим та зв'язним, буде кістяком цього графа. Цей кістяк буде мати мінімальну можливу суму ваг сукупності ребер, що означає загальну найменшу вагу дерева(кістяка).

Використання алгоритмів знаходження мінімального кістякового дерева допомагає у вирішенні багатьох різноманітних тривіальних та більш комплексних завдань. Наприклад розглянемо найбільш поширене завдання задля обчислення мінімального кістякового дерева: допустимо, що є n – кількість переправ, котрі потрібно з'єднати між собою шляхом так, щоб можна було досягти з будь-якої ділянки чи області в будь-який іншу місцевість, як безпосередньо, знайшовши найкоротший шлях, так і через обрані вузли(міста, ділянки чи області). В умові дозволяється будувати шляхи між вказаними парами переправ та надається вартість будівництва кожного такого шляху. Необхідно обчислити найбільш оптимальну кількість та розташування шляхій, котрі саме потрібно побудувати задля того, щоб зменшити до мінімуму загальну вартість будівництва.

Для вирішення цього завдання потрібно сформулювати задачу використовуючи терміни теорії графів, про знаходження мінімального кістякового дерева. Переправи будуть відповідати вершинам(вузлам) цього дерева, пари переправ, між котрими можливо прокласти прямий шлях, складають із себе ребра графу, а вартість будівництва відповідного шляху буде дорівнювати вазі ребра.

Із умов схарактеризованої задачі отримаємо граф, де G = (V,E), де V - це множина вершин, а E це множина ребер. І для кожного ребра $(u, v) \in E$ відома його вага $w(u, v)$. Мінімальним кістяковим деревом називається множина $T \subseteq E$, що поєднує всі вузли та повна вага буде найменшою.

$$\omega(T) = \sum_{(u,\vartheta)\ \epsilon\ T} \omega(u, \vartheta) \qquad (1.1)$$

За час розквіту комп’ютерних обчислень було винайдено достатньо різних алгоритмів вирішення завдань на пошук мінімального кістякового дерева. З них найбільш відомими та розповсюдженими можна зазначити:

- алгоритм Прима;
- алгоритм Краскала;
- алгоритм Борувки.

Розглянемо алгоритм Прима. Він має досить простий вигляд. Мінімальний кістяк, котрий ми за завданням шукаємо, буде будуватися поступово, за допомогою ребер, що додаються до нього по одному. Передусім кістяк буде складатися лише з одного вузла(вершини), котру можливо обрати довільно відповідно умов заданої задачі. Згодом обирається ребро з мінімальною вагою, котра виходить із заданого вузла(вершини), та додається в мінімальний кістяк. Після виконання цієї операції, кістяк буде мати в собі вже два вузла(вершини). Надалі відбувається пошук та додавання ребра з мінімальною вагою, що буде мати один свій кінець в одному з двох наданих візлів(вершин), а інший - навпроти, у всіх інших вузлах, за винятком цих двох. І так далі, тобто кожен раз, поступово, буде виконуватися пошук мінімального по вазі ребра, де один кінець якого – існуючий в кістяку вузол(вершина), а інший кінець - ще не існуючий вузол, що буде утворено на іншому кроці, і це ребро додається в кістяк, якщо таких ребер, що задовольняють умовам, кілька то можливо обрати будь-який із них. Цей процес повторюється до тих пір, поки кістяк не стане містити в собі усі вузли(вершини), або, інакше кажучи, n-1 кількість ребер.

В підсумку отримаємо побудований за умовами задачі кістяк, котрий буде мінімальним. Слід зазначити, що якщо заданий з самого початку в умові граф буде не зв'язний, то кістяк знайти не вийде, так як кількість обраних ребер залишиться менше ніж n-1.

Хоч алгоритм Прима і є найбільш відомим представником пошуку мінімального кістякового дерева, в сучасній галузі комп’ютерної графіки, де

важливу участь займають технології та обчислення в реальному часі(real-time technology), використовують інші алгоритми. Як більш доречний у цьому випадку приклад, розглянемо алгоритм A*(«A зірочка» або англ. «A star»).

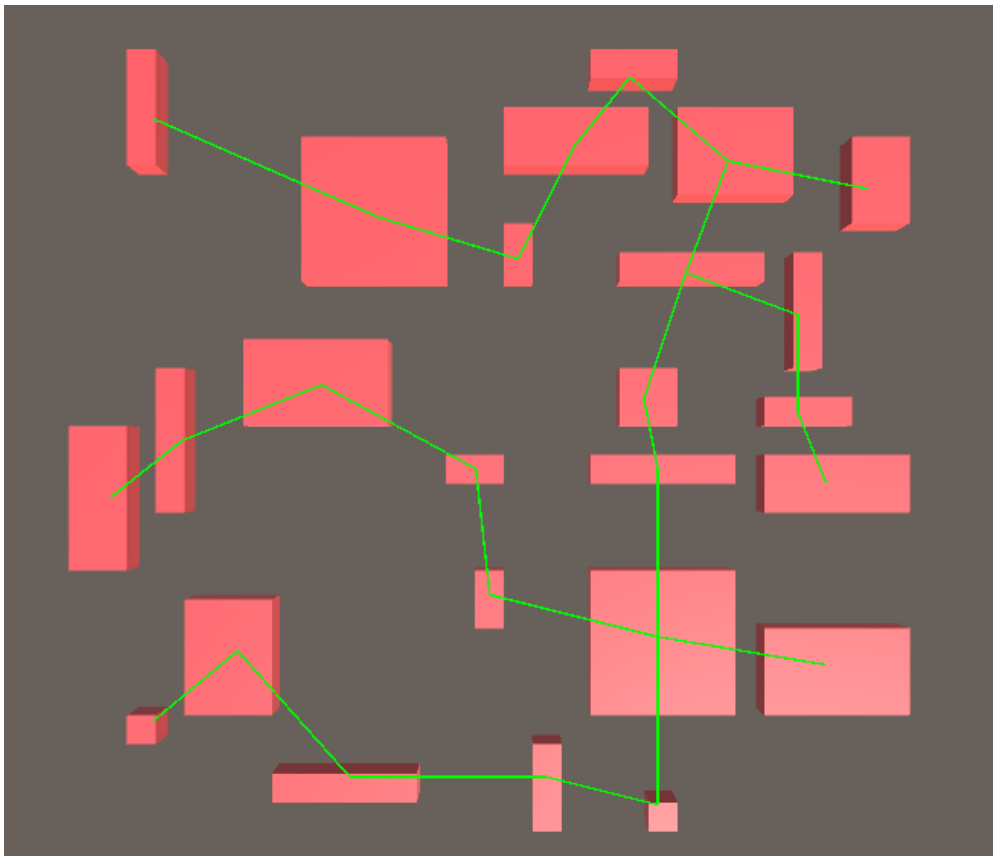

Рисунок 1.10 – Мінімальне кістякове дерево

Для кожного коридору в списку використаємо алгоритм A*, щоб знайти шляхи від початку коридору до кінця.

Сам алгоритм A* належить до евристичних алгоритмів пошуку, тобто внаслідок використання ми отримаємо найбільш оптимальний вихідний граф, проте, скоріш за все, не найкращий. Цей алгоритм застосовується задля пошуку найкоротшого шляху та дозволяє нам ставити пріоритети дослідження(вказувати вагу між двома вузлами(вершинами) графу).

Є модифікацією алгоритму пошуку Дейкстри та винайдений у своєму остаточному вигляді Пітером Хартом у 1968 році.

Таблиця 1.1 – Етапи модифікації алгоритму Дейкстри до досягнення

Ім'я винахідника	Розвиток	Ім'я алгоритму
Эдсгер Дейкстра 1959 рік	Основний алгоритм	Алгоритм Дейкстри
Нільс Нільссон 1964 рік	Створив евристичний метод для підвищення швидкодії алгоритму Дейкстри	A1
Бертрам Рафаель 1967 рік	Підготував основні удосконалення алгоритму і спробував уявити оптимальність, але зазнав невдачі	A2
Пітер Е. Харт 1968 рік	Представив диспут, який довів, що A2 є оптимальним при використанні узгодженої евристики з невеликою модифікацією	A*

Алгоритм Дейкстри досить вдалий у пошуку найкоротшого шляху, але він витрачає час на дослідження всіх напрямків, навіть безперспективних. Жадібний пошук досліджує перспективні напрямки, але може не знайти найкоротший шлях. Алгоритм A* навпроти, використовує як справжню відстань від початку, так і оцінену відстань до цілі. A* бере найкраще від двох алгоритмів. Оскільки евристика не оцінює відстані повторно, A* не використовує евристику для пошуку відповіді. Він знаходить оптимальний шлях, як алгоритм Дейкстри. A* використовує евристику(припущення) для зміни порядку вузлів, щоб підвищити ймовірність більш раннього знаходження вузла цілі.

Як вже було зазначено, цей алгоритм знаходить найменш витратний шлях до вузлу(вершини), який робить його одним із кращих алгоритмів пошуку. Основний принцип цього методу залежить від рівняння

f(x) = g(x) + h, де:

- x – вузол, або вершина,
- g(x) – загальна вартість від початкового вузла до будь-якого іншого x
- h(x) – еврестичне приближення, тобто припущення вартості шляху від поточного вузла(вершини) до цілі(кінцевий вузол),
- f(x) – мінімальна вартість шляху до сусіднього вузла.

Цей метод дозволяє шукачу дослідити шлях до цілі(і тільки до цілі), котрий допоможе уникати тих шляхів, що ведуть у глухий кут. Це робиться за допомогою створення двох списків "ЗАКРИТИЙ" і "ВІДКРИТИЙ". Застосування закритого та відкритого списків можна спостерігати у блок-схемі послідовності роботи алгоритму A*:

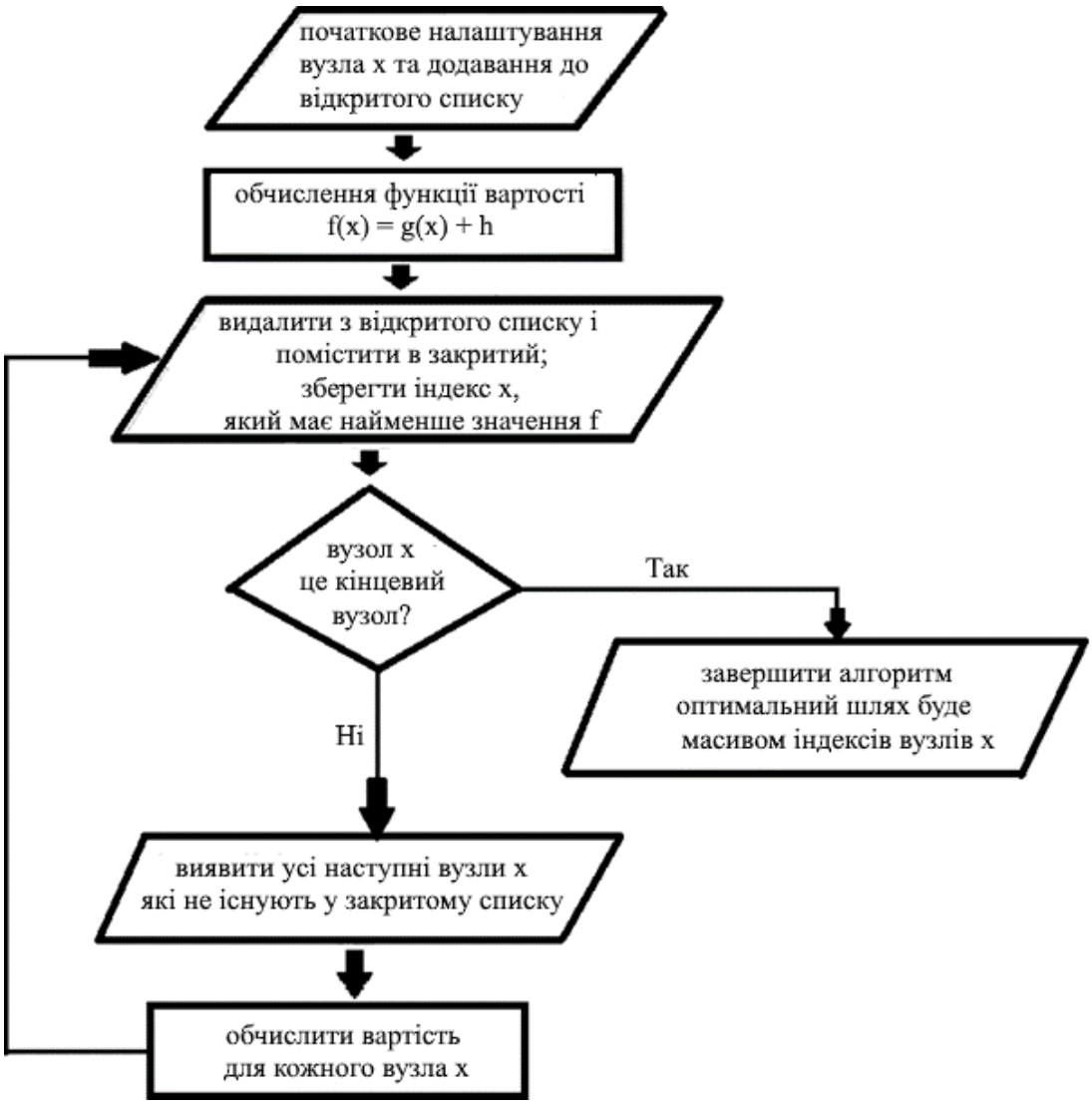

Рисунок 1.11 – Блок-схема роботи A* алгоритму

Закритий список використовується для запису і збереження вузлів, котрі були перевірені. Відкритий список, в свою чергу, використовується для запису суміжних вузлів(вершин) з уже розрахованими вартістю та відстаню, пройдених від початкового вузла(вершини) з відстанями до кінцевого(цільового) вузла(вершини), а також зберігає посилання на батьківський вузол кожного з обраних надалі вузлів. Ці батьківські вершини використовуються на останньому кроці алгоритму для планування шляху від мети(цілі) до початкового місця розташування вузла.

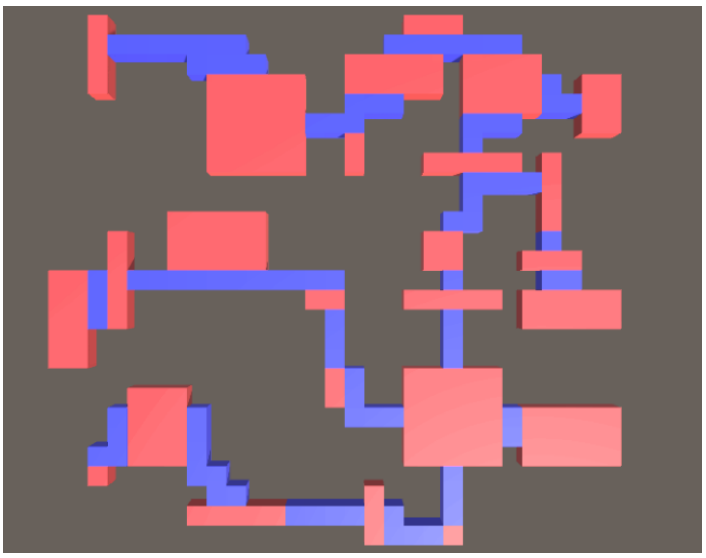

Рисунок 1.12 – Результат генерації за алгоритмом на основі BSP-дерева

1.3.2 Алгоритм «Drunkard walk»

Розглянуті алгоритми вирішують проблему пошуку оптимального шляху, однак у сфері обчислення комп'ютерної графіки є ще багато інших завдань, котрі необхідно розглянути. Наприклад процедурна генерація мапи. Існує багато методів генерації мапи, однак жоден з них неможливо вважати універсальним рішенням, тому більшість розробників підвлаштовують обрані ними алгоритми під свої потреби.

Одним із таких методів є алгоритм випадкове блукання або ще «Drunkard walk». Випадкове блукання відноситься до так званого математичного формалізму. Цей алгоритм має високу ступінь випадковості, що допомагає створювати дуже різноманітні відкриті та закриті простори.

Його основне призначення - опис траєкторії, що утворюється в результаті виконання послідовних випадкових кроків. Час від часу виникають цікаві випадкові блукання різного роду. Найчастіше розглядаються випадкові блукання, котрі мають в основі ланцюг Маркова, проте становлять інтерес також і інші складніші типи блукань.

Ланцюгом Маркова прийнято вважати це випадковий процес, що

задовольняє марківським властивостям і приймає кінцеве чи лічильне число значень (станів). Ланцюг може існувати як із дискретним, і з безперервним часом. Найпоширенішим способом візуалізації ланцюгів Маркова є графи переходів. Станами ланцюга називають вершини графа,і напрямне ребро проходить із вершини і у вершину j тільки у тому випадку, якщо ймовірність переходу між відповідними станами не дорівнює нулю. Ця можливість переходу також відображається поряд з відповідним рубом.

$$P = \begin{pmatrix} 1/2 & 0 & 1/2 & 0 & 0 \\ 1/4 & 1/2 & 1/4 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1/2 & 1/2 \end{pmatrix}$$

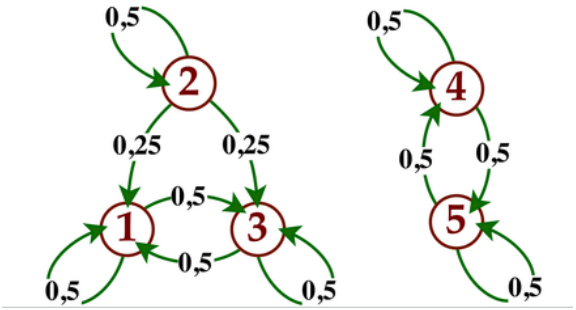

Рисунок 1.13 – Матриця ймовірностей переходу і граф переходів однорідного ланцюга Маркова з п'ятьма станами

Випадкові блукання можуть розглядатися в разніх розмірностях, іноді достатньо на прямій, або площині, можливе використання більш високих розмірностей також на групах. Випадкові блукання розрізняють і за відношенням до часового параметру. Хоча в більшості випадків випадкові блукання відбуваються у дискреному часі та для індексації використовують X0 , X1 , X2 , … Але деяким зник притаманне здійснення кроків у випадкові моменти часу ,тому в таких випадках координата Xt визначається на неперервному промені $t \geq 0$.

Розглянемо випадкове блукання.

$$Y_n = \sum_{i=1}^{n} X_i \text{ , де } EX_i = 0, DX_i = \sigma^2 < \infty \quad (1.2)$$

Згідно до центральна граничної теореми

$$\frac{Y_n}{\sigma\sqrt{n}} \to N(0,1), n \to \infty. \quad (1.3)$$

.

Але при випадкових блукання дане твердження можна значно підсилити.

Побудувши за Y_n випадковий процес

$$S_n(t), t \in [0,1], \quad (1.4)$$

та визначивши його так:

$$S_n\left(\frac{k}{n}\right) = \frac{Y_k}{\sigma\sqrt{n}}, \quad (1.5)$$

В випадку відмінних t потрібно довизначити процес лінійним продовженням:

$$S_n(t) = (k+1-nt)S_n\left(\frac{k}{n}\right) + (nt-k)S_n\left(\frac{k+1}{n}\right), t \in (\frac{k}{n}, \frac{k+1}{n}). \quad (1.6)$$

Відповідно до центральної граничної теореми маємо

$$\forall \mathrm{t}\, S_n(\mathrm{t}) \to N(0, \mathrm{t}), n \to \infty. \quad (1.7)$$

за розподілом. Звідси випливає збіжність одновимірних розподілів процесу $S_n(t)$ до одновимірних розподілів вінерівського процесу. Згідно до теореми

Донскера, відома як принцип інваріантності, необхідно пам'ятати про слабку збіжність процесів,

$$S_n(\mathrm{t}) \rightarrow W(\mathrm{t}), n \rightarrow \infty. \quad (1.8)$$

З поняття слабкої збіжності процесів маємо, що дана збіжність неперервна за вінерівською мірою функціоналів. Таким чином можна розраховувати значення функціоналів і від броунівського руху (максимум, мінімум, досягнення першого рівня, тощо) завдяки граничному переходу від простого випадкового блукання.

Хоча й стохастичну простоту алогритму можна вважати його недоліком, але Drunken Walk має безліч застосувань, як приклад для методу Монте-Карло до оцінки значень.

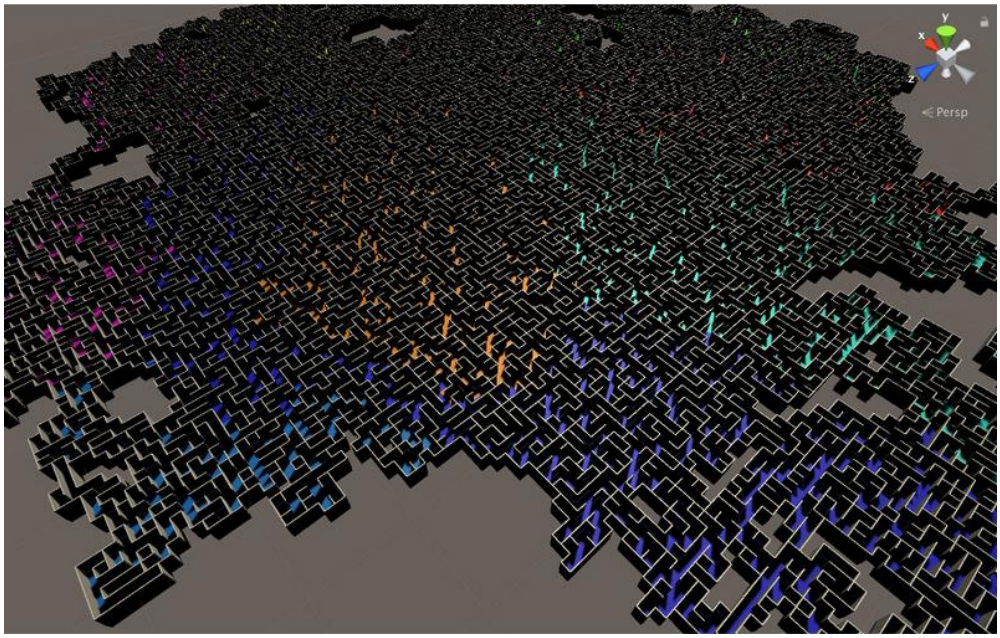

Рисунок 1.14 – Результат генерації за алгоритмом «Drunkard walk»

1.3.3 Клітинний автомат

В галузі комп’ютерної графіки реального часу існують такі завдання,

коли використання та обчислення графу може бути не оптимальним та витратним. В ситуаціях коли в умовах надається сітка у вигляді періодичної решітки, розраховувати за даними алгоритмами кожну клітину, вважаючи її вузлом безсумнівно не доцільно. Тому, в такому випадку, слід розглянути клітинний автомат та алгоритм що ґрунтується на ньому.

Автомат являє собою набір клітинок, котрі складають із себе певну періодичну решітку із деякими вказаними правилами переходу. Ці правила визначають, в наступний момент відліку, стан клітини через стан інших клітинок, що перебувають від цієї клітині на дистанції не більше вказаного, на теперішній момент відліку. Зазвичай, досліджуються ті автомати, де стан формується самою клітиною та її сусідніми клітинами.

На клітинному автоматі базується алгоритм, що розпочинає своє виконання з оголошення випадковим чином кожної точки рівня або «стіною», або «підлогою». Таким чином, виходить початковий стан клітинного автомата. Сусідами кожної з даних точок вважаються вісім інших суміжних до неї точок. Надалі, після призначення сусідних точок, розпочинається ітераційний процес, котрий окреслюється наступними правилами:

- якщо точка була до цього «стіною» та чотири її сусіди «стіни», то точка залишається «стіною»;
- якщо точка була «підлогою», проте п’ять її сусідів «стіни», то «стіною».
- Через декілька таких ітерацій, побудуються кімнати.

Однак, слід зауважити, що в результаті виконання алгоритму можуть формуватися ізольовані ділянки, котрих необхідно налагоджувати, як приклад, обробку проводити вручну або окремим алгоритмом.

Остаточний результат вміщує в собі порожні ділянки з пологими стінами, які природно виглядають та не мають строгого поділу на «кімнати» і «коридори». Такі рівні є добрим відображенням ландшафту природного середовища.

Саме досить нескладна побудова рівнів різноманітних та незвичайних форм є особливістю і перевагою цього алгоритму. В побудованій цим методом області не можна відслідкувати «кімнат» та «коридорів», як в попередніх двох розглянутих алгоритмах. Структура сформованої ділянки виходить згладженою і приємною на вигляд.

Проте існуюча відсутність гарантії зв'язності графа клітин становить головний недолік цього алгоритму.

Розглянемо способи вирішення зазначеної проблеми. Насамперед можна взяти будь-яку точку на рівні, запустити алгоритм заливки області та позначити всі вершини, що належать обраній ділянці. Надалі, всі інші точки, що не були помічені, оголошуються ізольованими від основного рівня і перетворюються в стіни.

Рисунок 1.15 – Результат генерації за алгоритмом клітинного автомату

Завдяки цьому поки число обраних алгоритмом вершин не буде більше вказаної константи, як приклад 0.5, що розділена на результат множення ширини і висоти всього простору, рівень не буде вважатися успішно

згенерованим, що вказує на те, що у випадку невдачі генерація рівня буде відбуватися знову, повністю спочатку, поки ця умова не виконається.

Роблячи перший огляд на обране рішення, можливо зробити висновок про відсутність оптимальності виправлень, однак, при великому розмірі простору, найчастіше на практиці спочатку обирається саме та точка, що належить найбільшій області. При малому ж розмірі простору ймовірність отримати незадовільний результат більш вища, проте генерація малих за площою рівнів потребує значно менше часу.

Друге можливе рішення проблеми включає в себе багаторазове використання алгоритму заливки, в наслідок чого буде побудувано безліч всіх можливих зв'язаних між собою ділянок на рівні. Далі за допомогою системи непересічних множин та, можливо, за допомогою алгоритмів пошуку найкоротшого шляху, окремі області поєднуються між собою коридорами. Вказаний процес вже більш комплексний та витрачає більше ресурсів. Окрім того, незважаючи на те, що цей недолік має вплив лише на зовнішній вигляд, прямі й рівні коридори можуть мати штучний характер серед «печер» незвичайних і випадкових форм, що формується за допомогою клітинного автомату.

Алгоритм має відносно низьку гнучкість. Серед параметрів, що можливо вказати лише ширину та висоту ділянки рівня та розподіл ймовірностей появи живих і мертвих клітин у першому поколінні. Ми маємо можливість модифікувати правила, згідно з якими живуть клітини в кожному поколінні, але при цьому досить важко передбачити та коректно налаштувати отриманий результат від таких змін. Можуть бути проблеми із визначенням та контролем площи рівня, тобто, в нашому випадку, кількість клітин «підлоги». Слід зауважити, що алгоритм породжує рівні виключно «печерного» виду. Такий підхід часто використовується у підгалузі комп’ютерної графіки, а саме, у відео-іграх. Проте, хоч для деяких представників це є бажаним результатом, однак для інших, де місце дії чи обраний дизайн повиннен мати вигляд інтер'єру, розглянутий алгоритм

принципово не підійде.

1.3.4 Пагорбний алгоритм

Пагорбний алгоритм (Hill Algoritm) вважається одним з найпростіших алгоритмів генерації ландшафту. В своїй основі це простий ітераційний алгоритм, заснований на декількох вхідних параметрах. Основною ж особливістю є початок з неоптимального рішення, яке покращується доки результат не будет відповідати поставленній умові. Для цього достатньо виконати декілька кроків:

1) Відбувається створення двовимірного масиву ініціалізованого нульвоим рівнем, тобто заповненим нулями;

2) Обирається випадкова точка ландшафкту, чи за його межами (біля кордону), а також випадковий радіус за встановленими межами, дані межі обираються згідно до потреб, бо саме від них буде залежити вид ланшафте: пологий, скелястий.

3)Згідно до обраної точки застосовуємо обрані дані та «піднімаємо» пагорб заданого радіуса;

Більш детальніше розберемо даний шаг. Пагорб з матиматичного боку схожий на параболу, а отже для генерації пагорбів можна використати формулу для поодиночної генерації :

$$z = r^2 - ((x_2 x_1)^2 + (y_2 - y_1)^2). \quad (1.9)$$

Де (x1, y1) — це задана точка, r — вибраний радіус , а (x2, y2) — висота пагорбу. В результаті маємо:

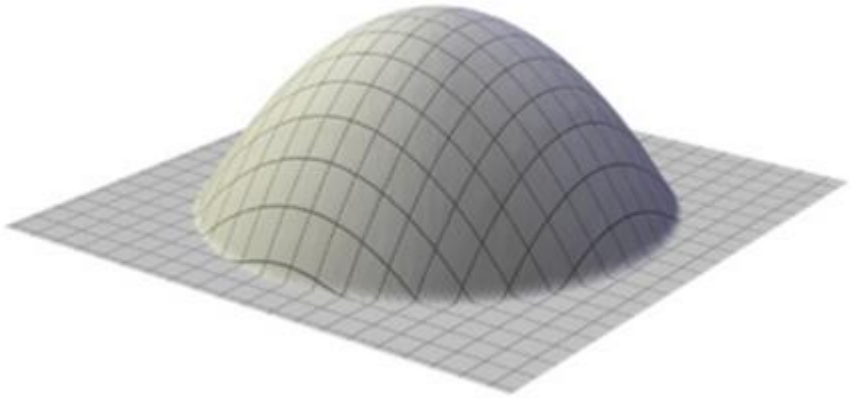

Рисунок 1.16 – Приклад згенерованого одного з пагорбів

4)Повторюємо всі дії з другого кроку, доки не досягнемо встановленій на початку кількості кроків.

Безпосередньо, чим більша кількість кроків була обрана тим більше буде змінено зовнішній вигляд нашого ландшафту.

В результаті побудування множини пагорбів бажаний ландшафт буде сгенеровано, але є дві умови до яких потрібно приділити увагу.

По-перше це ігнорування негативних значень висоти пагорба. По-друге рекомундується, що отриманне значення потрібно додавати до вже утворенних значень це покращить генерацію наступних пагорбів. Це надає можливість до побудування більш правдоподібного ландшафту, ніж правильно окреслені округлі пагорби.

Подивіться, як виглядає ландшафт при великій кількості ітерацій:

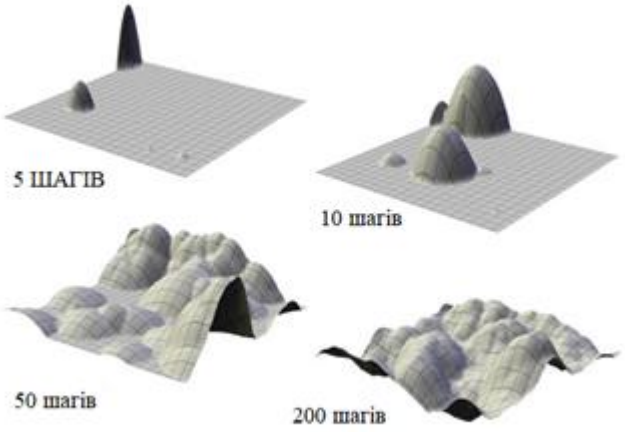

Рисунок 1.17 – Результат генерації холмів в декілька шагів

1.3.5 Алгоритм Diamond-Square

Алгоритм Diamond-Square – це метод створення карт висот для комп'ютерної графіки. Вперше ця ідея була представлена Фурньє, Фасселом і Карпентером на конференції siggraph у 1982 р. Потім її проаналізував Гевін Міллер на конференції siggraph у 1986 р. лише через чотири роки.

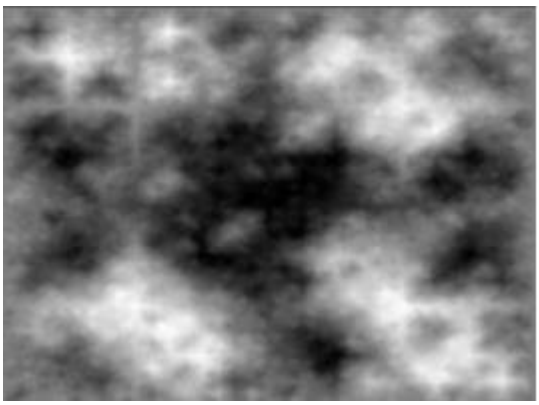

Рисунок 1.18 – Результат генерації за алгоритмом Diamond-Square

Почтаковими даними для алгоритму diamond-square є двовимірний масив за розміром 2n + 1. У чотирьох кутових точках масиву встановлюються початкові значення висот. Усього існує два кроки котрі повторіються доки всі

значення масиву не будуть встановлені:

- крок diamond ромб. Для кожного ромба в масиві, встановлюється серединна точка, якої присвоюється середнє арифметичне з чотирьох кутових точок плюс випадкове значення;

- крок square квадрат. Для кожного квадрата в масиві, знаходиться серединна точка, в яку встановлюється середнє значення чотирьох кутових точок плюс випадкове значення.

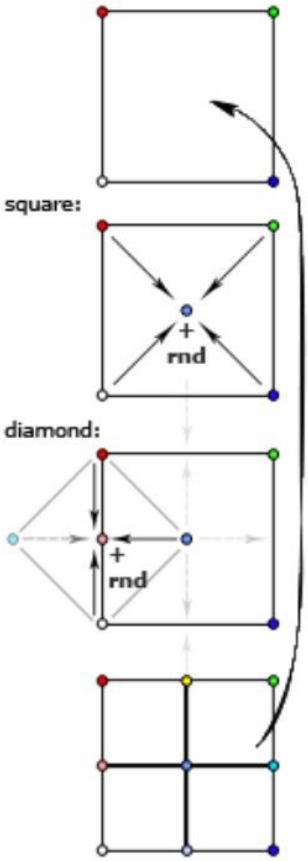

Рисунок 1.19 – Графічне пояснення кроків при використанні алгоритму Diamond-Square

Випадкове число обирається згідно до встановленого проміжку, але зазвичай фактор нерівності дорівнює значенню проміжку від 0 до 1, а і це номер , де пара кроків вважають однією ітерацією. Таким чином при кожній ітерації випадкове значення буде зменшуватись. Але в кроках diamond є особливість, адже по краях загального масиву будуть мати тільки три сусідніх значення , пов’язано це з тим що четверта точка буде виходити за розмірність масиву. Однак існує декілька способів для вирішення поставленої проблеми, серед яких просто взяти середнє від трьох крайніх точок. При послідовному використанні із зазначенням загальних початкових значень, цей метод дозволяє "зшивати" генеруються карти висот.

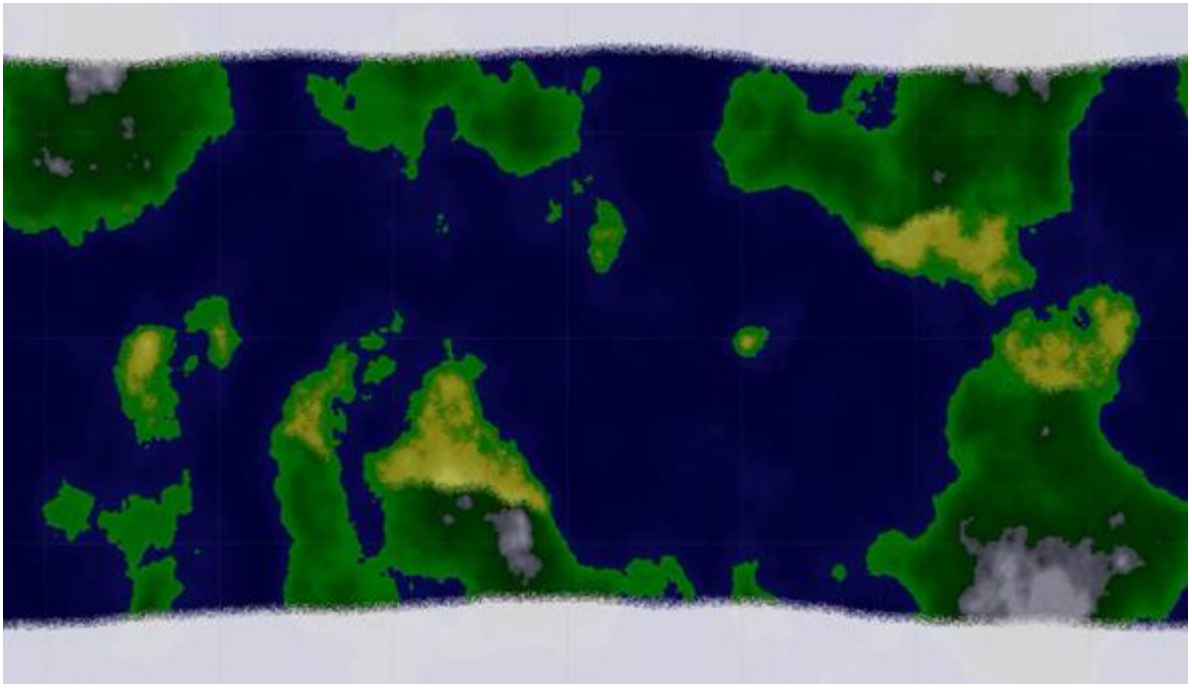

Рисунок 1.20 – Результат генерації за алгоритмом Diamond-Square

1.3.6 Алгоритм на основі Шуму Перліна

Perlin Noise – це математичний алгоритм, в основі якого використовується однойменна шумова функція вперше про яку було написано у 1985 році Кеном Перліном в Siggraph під назвою «Синтезатор зображень». Дана шумова функція була значно вдосконалена та розширена за багато років першочергово автором та іншими.

Шум Перліна має більш органічний вигляд, оскільки він створює природно впорядковану («плавну») послідовність псевдовипадкових чисел. Нижче показано шум Перліна з часом, вісь x представляє час; зверніть увагу на плавність кривої (рисунок 1.19).

Рисунок 1.21 – Шум Перліна з часом

На відміну від цього, наступний (рисунок 1.20) нижче показує чисті випадкові числа з часом.

Рисунок 1.22 – Випадкові числа з часом.

Взагалі шум Перліна реалізують як дво-, три-, чотиривимірна функція, при цьому доступна довільна кількістю вимірів. А для виконання виділяють такі кроки:

- визначення сітки;
- обчислення скалярного добудку градієнтних векторів;
- інтерполяція між цими значеннями.

Розберемо дані кроки детальніше. Спочатку визначається n-вимірна сітка, надалі до кожної координати сітки відбувається присвоєння n-вимірного одиничного вектору. Якщо розглядається одновимірна сітка, то

кожній координаті буде присвоєно значення +1 або -1, у випадку з двовимірною сіткою маємо, що всім координатам присвоюється випадковий вектор з одиничного кола, при збільшенні кількості вимірів ідентично збільшуємо.

При визначенні випадкових градієнтів у більшій кількості вимірів використовується наближення Монте Карло, згідно с чого з одиничного куба обираються випадкові Картезіанскі кординати, надалі ж цей не припиняється пока не буде дотягнута потрібна кількість випадкових градієнтів, після чого вектори нормалізуємо.

Також можливе використання хеш-таблиць з таблицями пошуку, що приводить до зменшення витрат на обчислення градієнтів для кожної з кординат. Додатково з'являється можливість додаваннся випадкових сідів. У випалку використання сідів, достатнью лише. кілька разів застосувати шум Перліна.

Переходяи но наступного кроку визначається комірка сітки х потраплянням туди точки. Для кожного вузла сітки визначаємо вектор відстані між точкою і координатами вузла. Далі обчислюємо скалярні добутки векторів відстані та градієнтних векторів кожного вузла комірки.

Для кожної точки у двовимірній сітці процес вимагатиме 4 операції, у тривимірній відмовідно маємо 8, відповідно отрим, що складність складає $O(2^n)$.

В останню чергу визначається інтерполяція значень скалярних добутків, обчислених для кожного вузла. Інтерполяція виконується з використанням функції, що має нульову першу похідну (і, можливо, другу похідну також) на обох кінцевих точках. Лінійна функція, для кінцевих точок на 0 та 1, зі значеннями a0 та a1, може бути такою:

$$f(x) = a_0\,(1 - x) + \,a_1 x \tag{1.10}$$

Загалом фунція шуму дорівнятиме значенням від -1 до 1 у

комп'ютерній графіці, але можливі виключення, для цього потрібно використовувати масштабуючі множники, аби результати шуму Перліна залишилися у проміжку.

1.3.7 Алгоритм Форчуна

Алгоритм Форчуна представляє собою алгоритм планарного замітання для множини точок за час $O(n \log n)$ із використанням $O(n)$ простору. Перша публікація і сама назва була отримана в честь Стіва Форчуна у статті «Алгоритм лінійної розгортки для діаграми Вороного».

Діаграмою Вороного використовується для опису просторового відношення між близькими, чи найближчими для точок сусудів. Названа в честь українського математика Георгія Вороного, в своїй основі цей спосіб представляє поділ простору на певні області (комірки), а також вхідними точкам (ділянками). Умовою є те, що комірка містить лише одну ділянку, а точки у комірці розміщені посередині одночасно найближче розміщені в середині цієї комірки.

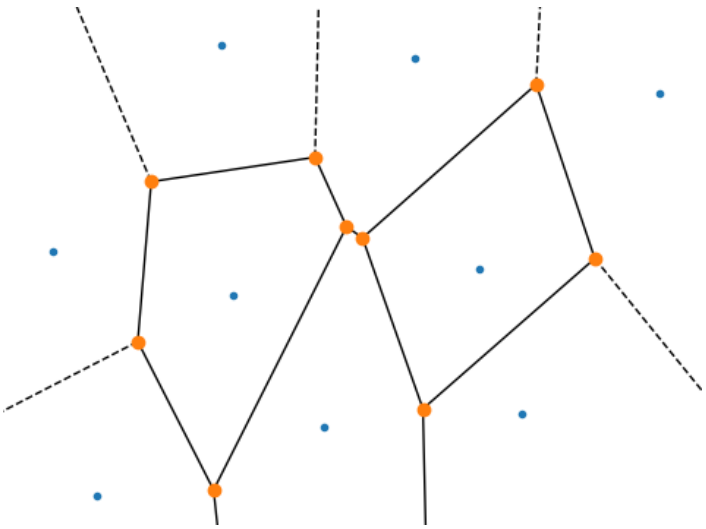

Рисунок 1.23 – Випадкові числа з часом

Діаграми Вороного давно широко застосовуються у комп’ютероній

графіці та моделюванні: створення текстур води, генерація об'єктів для кам'янистого ґрунту, керування руху агентів тощо .

Повертаючись до алгоритму, особливістю даного алгоритму є підхід розгортки. Почергово роглядаються ділянки так збільшується комірка навколо кожної з ділянок, таким чином щоб вона була оточена клітинами і вже не могла збільшуватись. Саме тому цей алгоритм називають алгоритмов «зачистки» площини.

Таким чином алгоритм виглядає так, якщо існує певна лінія L, яка рухається зверху вниз і вона проходить через кожну задану точку, прицьому дані точки вже належать діаграмі Вороного, то маючи точку x, що знаходиться на однаковій відстані від точок a та b (тобто всі вони лежать під лінією розгортки) розглядана точка не обов’язково має бути на діаграмі, адже над лінією розгортки може знаходитись наступна ділянка, котра ближче. А оскільки визначення діаграми Вороного можливе лише для точок, розтанованих ближче до буль-яких інших під L, то все ці точки лежать під параболою, а точка a – це фокусна точка, лінія ж розгортки тоді директиса згідно визначенню параболи.

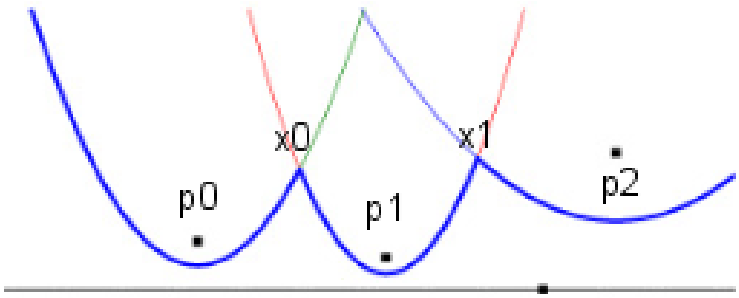

Рисунок 1.24 – Приклад пляжної лінії

Таким чином формується пляжна лінія – це послідовність дуг p і перетинів x, що називають ребрами, бо формують ребро графа Вороного, границя згідно до якої позначаються точки до яких можливо намалювати

діаграму і складається з дуг парабол, перетини ж дуг у парабол з однаковими відстанями до якихось двох ділянок (двох фокусів парабол) та до лінії розгортки. В результаті при при русі лінії відображається діаграма. Пляжна лінія - $p_0, x_0, p_1, x_1, \dots, x_{n-1}, p_n$.

В результаті алгоритму маємо:

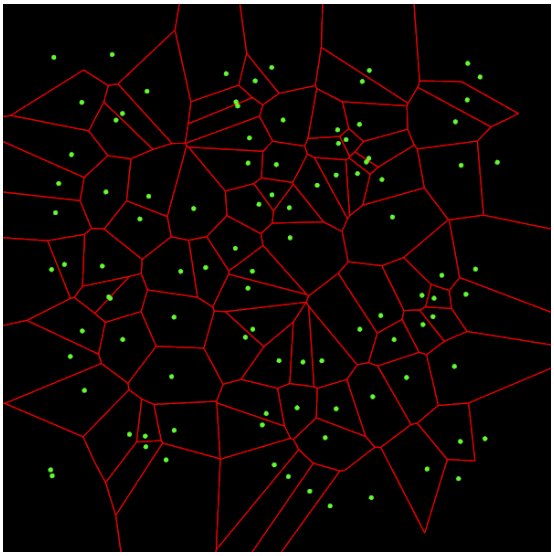

Рисунок 1.25- Результат алгоритму Форчуна

2 АНАЛІЗ ДОСТУПНИХ ЗАСОБІВ ПРОГРАМУВАННЯ

Кваліфікаційна работа написана на мові програмування C#/.NET з використанням середовища Unity та JetBrains Rider.

Unity було розроблено американською компанією Unity Technologies та випущено у 2005 році. Вже багато років посягає місце одного з лідерів серед середовищ розробки комп'ютерних ігор, а можливість до міжплатформності підігріває ще більше інтерес користувачів, кількість платформ лише з кожним роком тільки збільшується та досягла вже понад 20, що включають персональні комп'ютери, ігрові консолі, мобільні пристрої, інтернет-додатки та інші. Середовище постійно покращує свій функціонал та з кожним роком все більш активно демонструє новинки.

Вже протягом багатьох років постійно відбувається випуск нових ігор, додатків, візуалізації математичних моделей, які охоплюють безліч платформ і жанрів, а кількість перевищує десятки тисяч. Функціонал та простота у використанні відіграють значну роль, саме тому Unity використовується як великими розробниками, так і незалежними студіями.

Здебільшого ігровий движок покриває весь спектр можливих потреб у функціоналі для використання в різноманітних іграх, серед яких моделювання фізичних середових, динамічні тіні, карти нормалей та багато іншого. До особливостей, що відірізняють даний ігровий движок від інших це візуальне середовище розробки та міжплатформна підтримка.

Щодо першої особливості, то основною різницею від інших є не лише простий інструментарій візуального моделювання, а й інтегрована середа, лінію складання, що значно покращує продуктивність та зменшує час та складність при створенні можливих прототипів та тестових елементів.

Під міжплатформеність мається на увазі, що підтримується не просте розгортання згідно з встановленної платформи (персональні комп'ютери, мобільні пристрої, консолі і т. Д.), Але і наявність інструментарію розробки

(інтегроване середовище може використовуватися під Windows і Mac OS).

Головною особливістю та перевагою Unity постає модульна система компонентів (Entity-Component Architecure), завдяки якої відюувається конструювання ігрових об'єктів - сцен (Scene), які заповнені налагодженими згідно з умови об’єктами (GameObject). Кожен об’єкт може мати персональний список компонентів (Component) , а отже і персональний список дій та загальні:

-Transform – контролює положення об’єкту у просторі;

-Collider – відповідає за обробку дотиків;

-Rigidbody – відповідає за фізичні характеристики.

Додатково у Unity присутній особливий компонент дозволяючий додавати до компонентів додаткові персонально розроблені компоненти MonoBehaviour. Відповідно розробнику потрібно лише зробити нащадка від даного класу і додати потрібний функціонал.

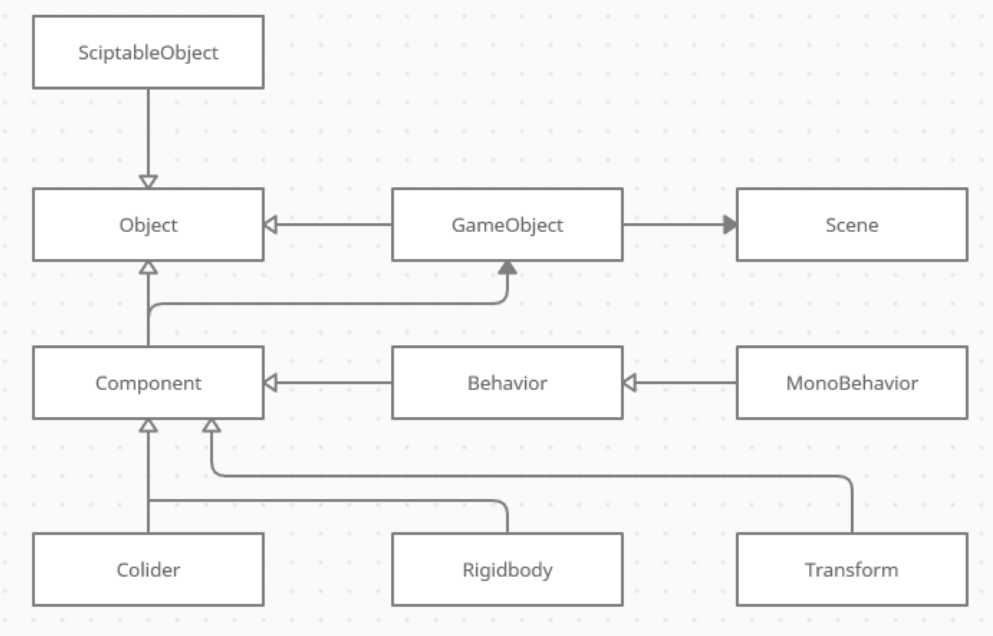

Рисунок 2.1 — архітектура Unity

Важливо відзначити об’єкт ScriptableObject, з метою покращення продуктивності та зменшенню навантаженню було додано об’єкт , що має

значно простий цикл життя та для користування потребує до створення спеціального файлу, здебільшого використовується як контейнер відповідних даних. Додатково можна задавати складні стани згідно до функціоналу редактора.

При завантаженні гри ці об'єкти десеріалізуються з файлів у повноцінні об'єкти системи, мають свою функціональність. Основна причина використання таких об'єктів є те, що вони не знищуються збирачем сміття Unity при роботі системи, у той час як об'єкти звичайних класів(не успадкованих від ScriptableObject або MonoBehaviour) будуть.

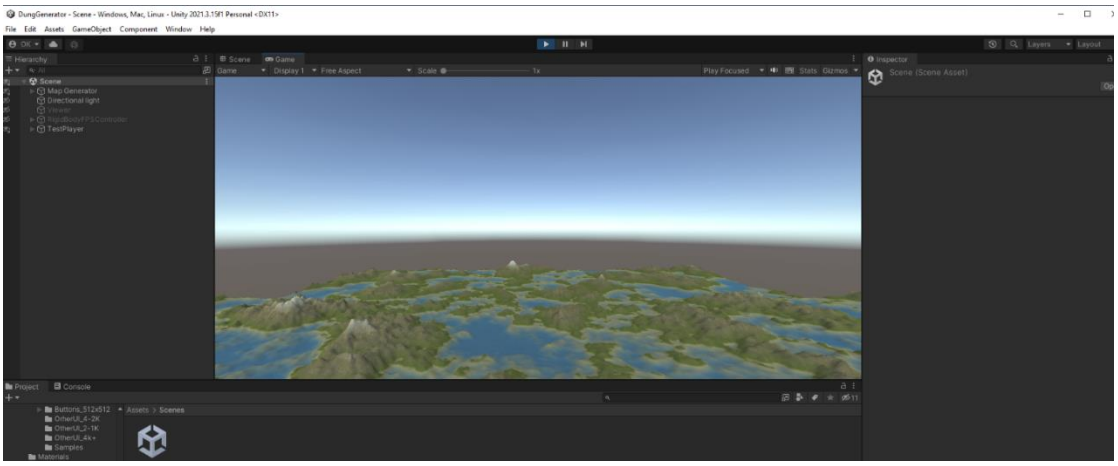

Рисунок 2.2 — графічний інтерфейс Unity

Але присутні і недоліки, яке приклда це обмеження візуального редактора при роботі з багатокомпонентними схемами. До другого недоліку відзначають відсутність підтримки Unity посилань на зовнішні бібліотеки, через це при розробці додатково потрібно самостійно їх налаштовувати, а через це розробка може значно бути довшою та складнішою. Останнім недоліком можно назвати користування шаблонними примірниками (Prefabs). З одного боку, ця концепція Unity пропонує гнучкий підхід візуального редагування об'єктів, але з іншого боку, редагування таких шаблонів є складним і вирішення проблем при одночасному їх редагуванні може зовсім зламати об’єкт.

JetBrains Rider називаться інтегрована середа розробки (IDE), що за останній роки все більше набирає популярності серед розробників, а саме C#//.Net. Як і іншим IDE до його особливостей входить об’єднання великої кількості функціоналу інструментів, як приклад текстовий редактор, або компілятор для спрощення розробки. Він легко використовується при написанні коду, тестуванні, аналізу якості та продуктивності коду. Доступний же він на всіх популярних платформах: Windows, Mac і Linux.

Для Rider немає різниці в типі розробляємого додатку, чи це простий додадток до магазину, гра, сервер, або велика складна система для обслуговування підприємста та обробки великої кількості данних. Таким чином розробникам надається можливість до:

створення додатків та ігор;

веб-сайти і веб-служби на основі ASP.NET, JQuery, AngularJS і інших популярних платформ;

ігри і графічні додатки для різних пристроїв;

Rider поєднує можливості ReSharper в частині аналізу .NET-коду з функціональністю IntelliJ-платформи. Наприклад, в Rider є більшість можливостей WebStorm для розробки фронтенда і DataGrip для написання SQL і роботи з базами даних.

Ще однією з важливих особливостей Rider виступає дуже глибокий аналіз коду. Розробнику надається понад 2000 інспекцій, що допомогають виявляти та аналізувати помилки, присутнє автоматичне виправлення.

Додадково доступна технологія IntelliSense і можливістю найпростішого рефакторінгу коду. Присутній вбудований відладчик працюючий на двух рівнях, як вихідного коду так і машинного. Інші ж інструменти додані для спрощення створення графічного інтерфейсу у додатках, дизайну схем баз даних тощо. Відсутні обмеження щодо сторонніх додатків(плагінів) розробник може легко розшири фунціональність на кожному рівні, будь то контроль версій, чи специфічного аналізу та функціоналу згідно з потреби створеного проекту.

3 ОПИС АЛГОРИТМІВ

Розглянемо алгоритм, розроблений в рамках даної кваліфікаційної работи. З його допомогою процедурно генеруються трьохвимірні ігрові рівні, подання яких засновано на заготовлених кімнатах та ландшафті. Основна ідея алгоритму передбачає створення об'єктів помодульно, згідно з поставленої задачі.

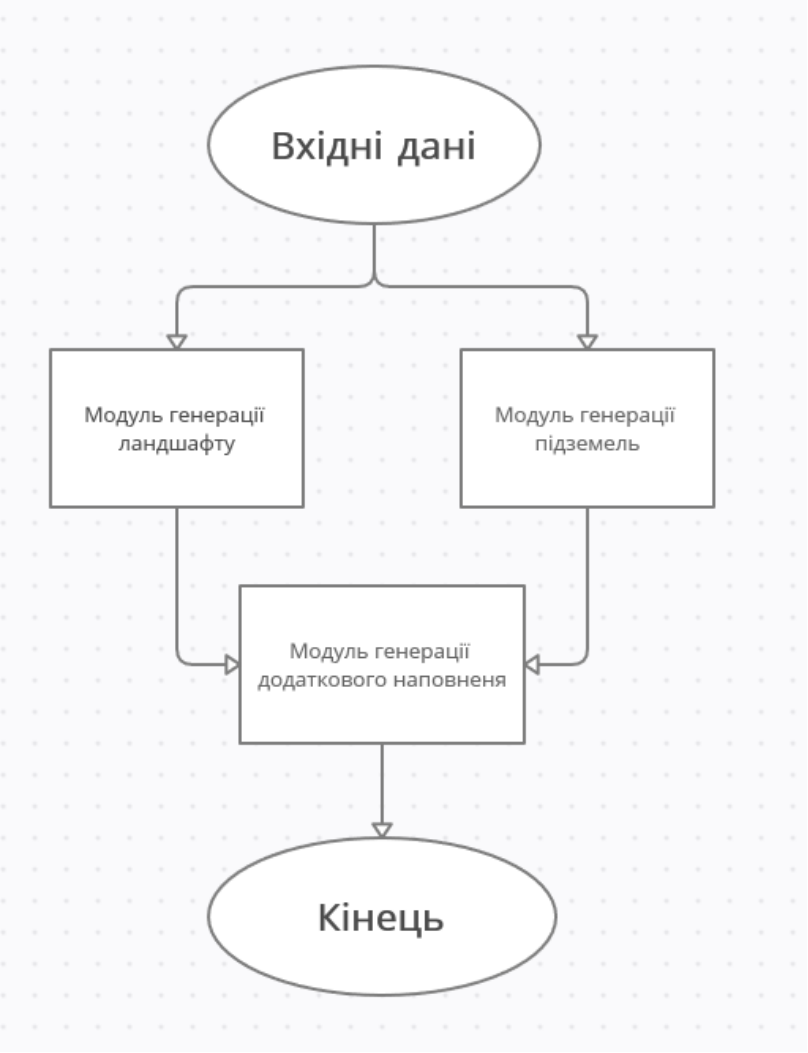

Рисунок 3.1 — графічний інтерфейс Unity

3.1 Модуль генерації ландшафту

Для генерації ж ландшафту був використаний шум Перліна , вже детально описаний в даній роботі. Вхідні параметри:

- розмір ландшафту – обовз’язковий параметр,бо саме він задає бажані для отримання розміри згенерованого об’єкту;
- крива висот - він задає коэфієнти ,які впливають на висоту точок у сгенерованій матриці;
- кількість октав – використовується для отримання найбільш задовільного результату при генерації.

Етапи генерування:

1) відбувається підрахунок зміщення октав;

2) надалі згідно з введеними розмірами площини та кількості октав рахуються можливі кординати для створеного ландшафту;

3) знаходимо середнє значення відносно індексу в отриманому списку можливих кординатів ландафшу;

4) отримані кординати згідно з кривої висот переносяться на графічний матеріал;

5) згідно з відношення мінімальной та максимальної висоти згеренованого об’єкту йому надаются зазделегіть підготовлені кольори.

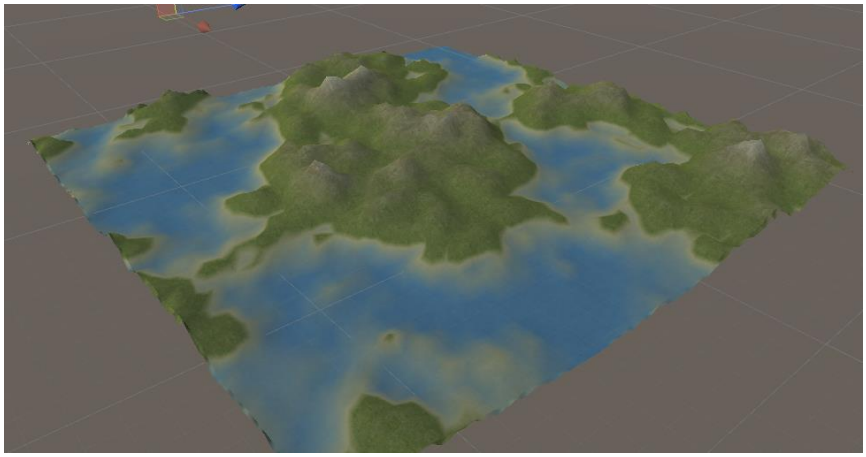

Рисунок 3.2 – Приклад згенерованого ландшафту

3.2 Модуль генерації підземель

Особливістю алгоритму виступає те,що при генерації кімнати візуальні елементи ніяк не впливають на кординати та на фінальний розмір згенерованих кімнат. При створенні кімнат створюються додаткові вокселі.

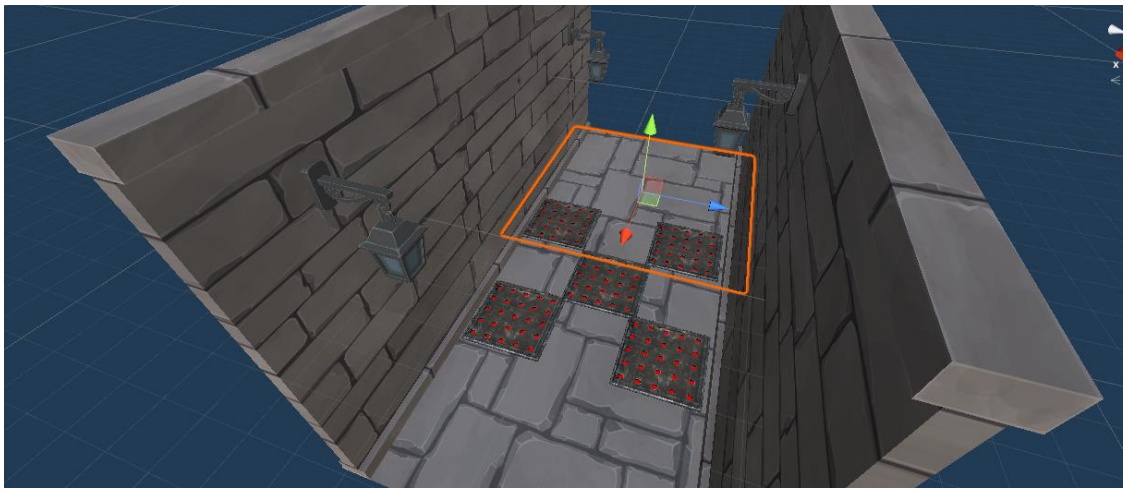

Рисунок 3.3 – Приклад кімнати

Воксель - це елемент простору, що представляє певну кількість значень у комірці однорідної просторової сітки. Подібно до пікселів у двомірному зображенні (значення осередків сітки — це кольори). Вокселі традиційно використовуються для візуалізації та аналізу медичних та наукових даних. Крім того, передові програми 3D-моделювання та комп'ютерні ігри використовують цю технологію для створення складного ландшафту, включаючи арки та печери, які неможливо визначити за допомогою карт висот.

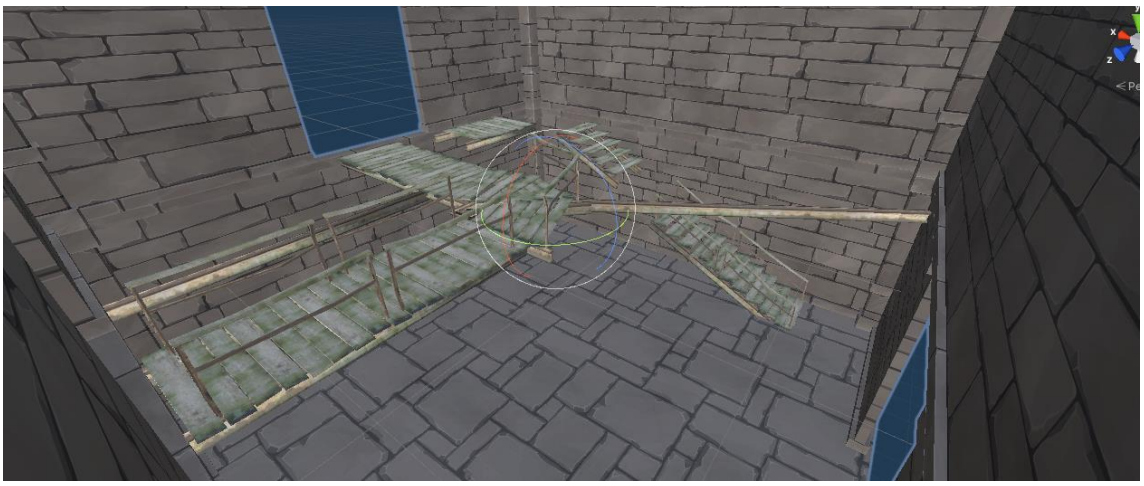

Рисунок 3.4 – Приклад кімнати 2

Подібно до пікселів, самі вокселі, як правило, не містять інформації про своє розміщення в просторі (координати), натомість координати обчислюються на основі розташування в структурі даних, яка утворює єдине просторове зображення.

Саме завдяки їх особливості при генерації лабіринту враховуються лише їх кординати.

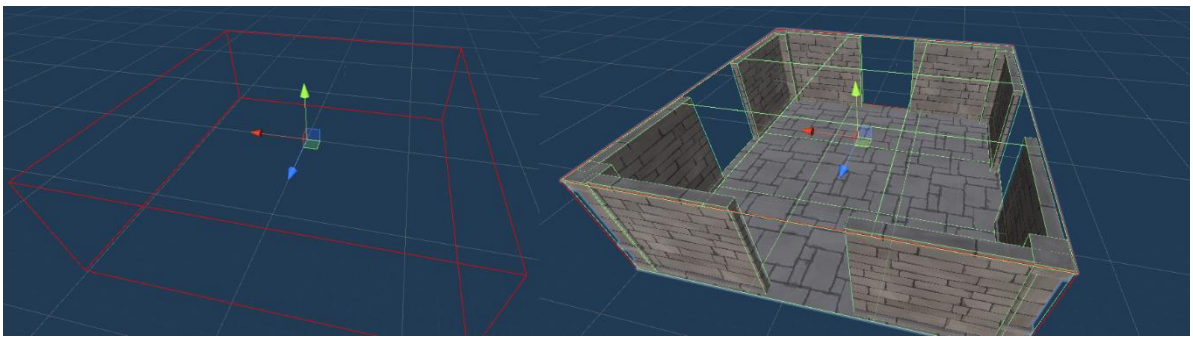

Рисунок 3.5 – Демонстрація вокселів

Вхідні параметри:

- кількість кімнат – обовз’язковий параметр,бо саме він задає довжину підземелля;
- набір кімнат – зазделегіть підготовлені кімнати з своїм персональним

наповненням та внутрішніми генераторами.

Для спрощення опису алгоритму будемо вважати, що всі розглянуті параметри доступні з будь-якої точки алгоритму. У багатьох мовах програмування цього легко досягти, оголосивши такі змінні, як поля одного класу, або використовуючи глобальні змінні. Також реалізації алгоритму не потрібні рішення, але у разі значно скорочується передача параметрів між методами алгоритму.

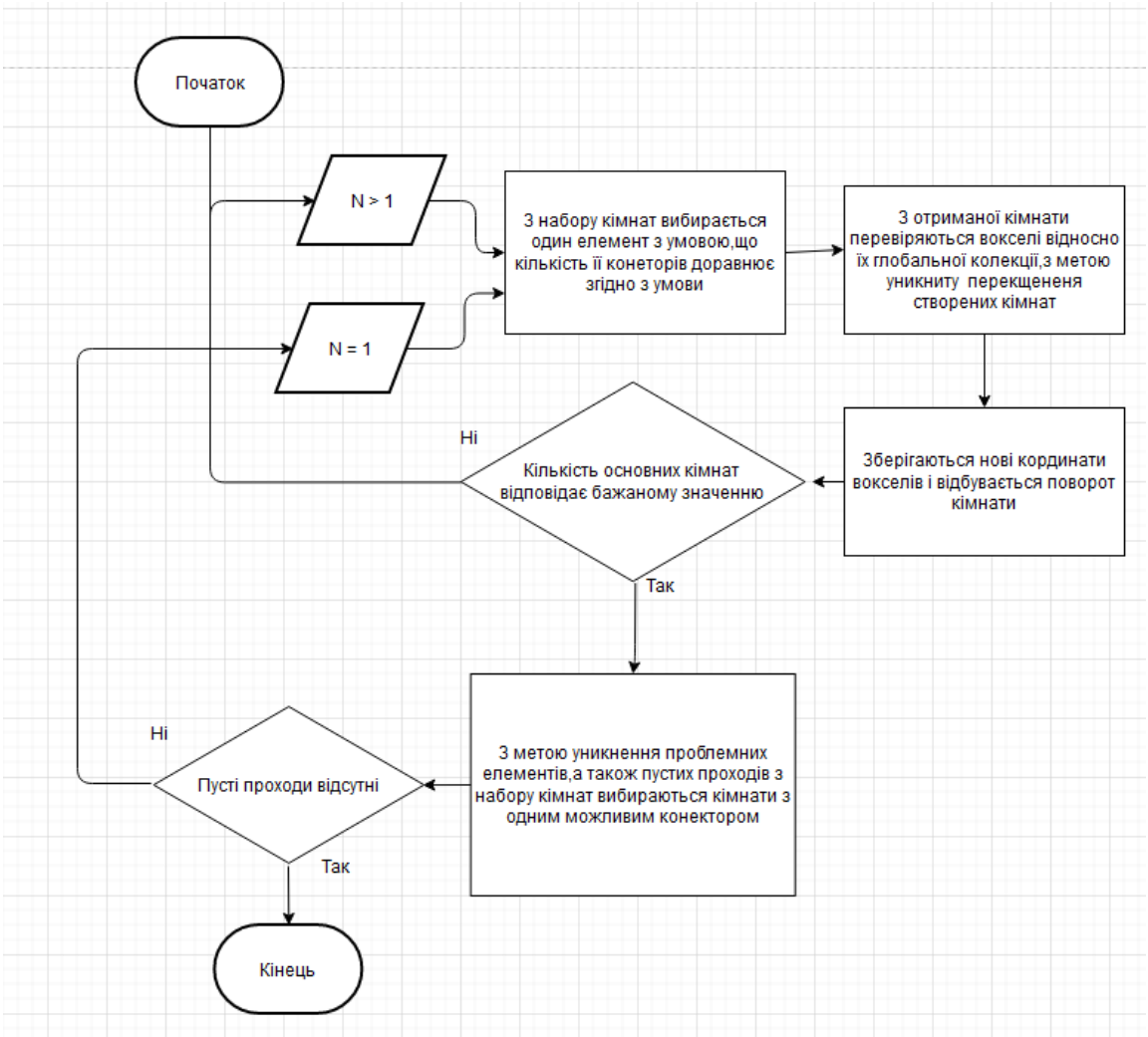

Рисунок 3.6 – Блок-схема розробленого алгоритму

Етапи генерування:

1) отримується перша кімната – початок відносно якого будуть створюватись нові сусіди;

2) з набору кімнат вибирається один елемент з умовою,що кількість її конеторів більше одного;

4) ж отриманої кімнати перевіряються вокселі відносно їх глобальної колекції,з метою уникниту перекщененя створених кімнат;

5) зберігаються нові кординати вокселів і відбувається поворот кімнати;

6) пункти 2-5 повторються доки кількість створених кімнат не буде; відповидати кількості відповідно при старті генерації;

7) з метою уникнення проблемних елементів,а також пустих проходів с набору кімнат вибираються кімнати з одним можливим конектором та повторються пунки 3-5 доки всі можливі кімнати не отримаються сусідів.

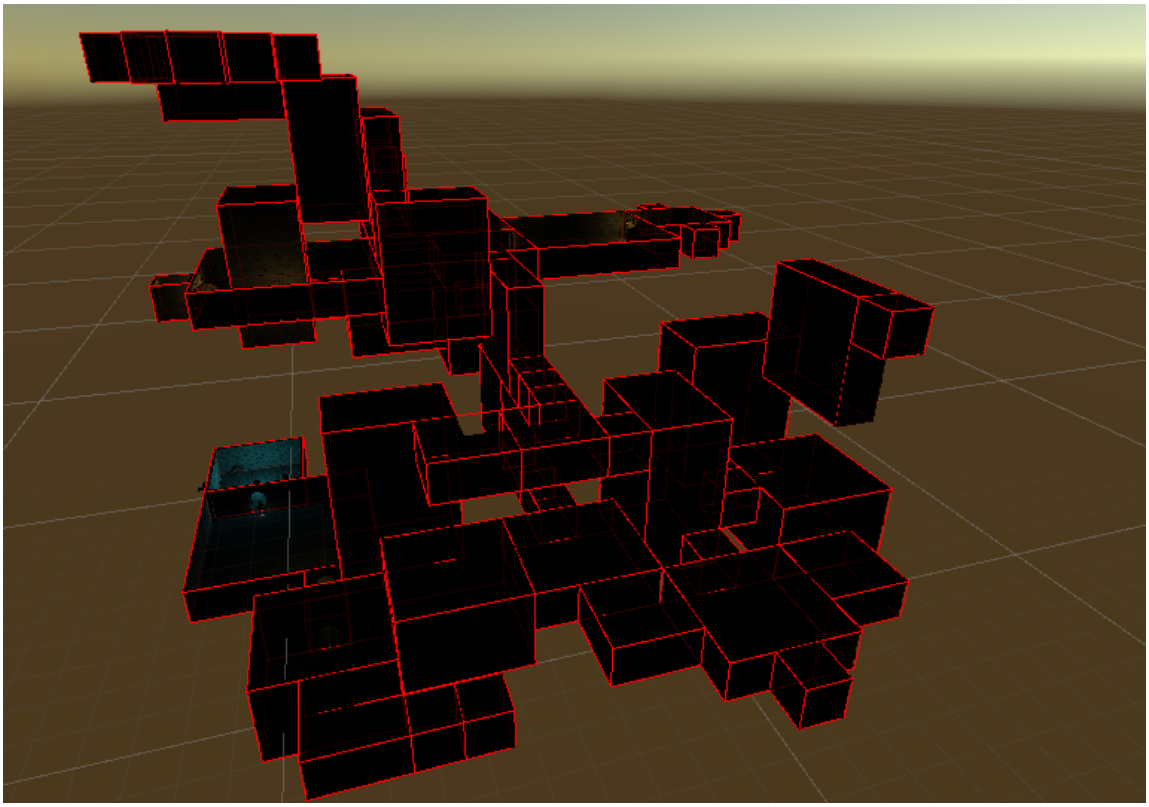

Рисунок 3.7 – Згенероване підземелля з контуром кімнат

Використовуючи цей алгоритм, ви можете створити підземелля не тільки на одному рівні, але й з певним так званим підйомом, який показано

на рисунку 3.7.

3.3 Модуль генерації додаткового наповнення

Наповнення згенерованого світу є дуже важливою частиною, бо саме завдяки цьому створений світ вважається «живим».

Основною задачою було зв’язати утворені підземелля з світом, тому було вирішено додати певні об’єкти(портали) для взаємодії. Для цього було використані вхідні дані такі як ліміт кількості порталів, який відповідно дорівнює кількості створених підземель, а також карта висот завдяки якій був побудований ландшафт.

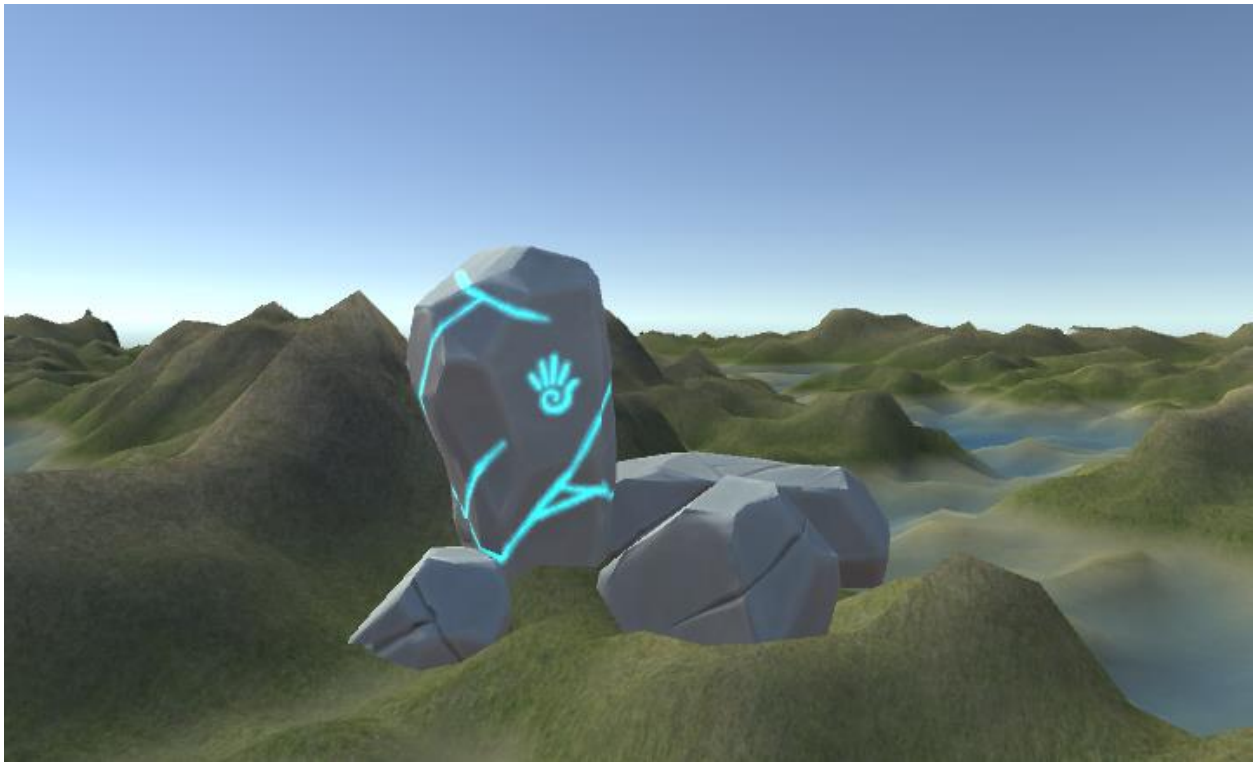

Рисунок 3.8 – Приклад згенерованого порталу

Додатково було вирішено наповнити світ рослинами, але в даному випадку були використані особливості Unity, а саме клас Collider. Для спрощення логіки генерування для кожного блоку карти була створена додаткова площина до якої був доданий даний компонент. Після чого

отримувались точки перетину, які надалі були використані при проекції для встановлення точки генерації у мірках всесвіту.

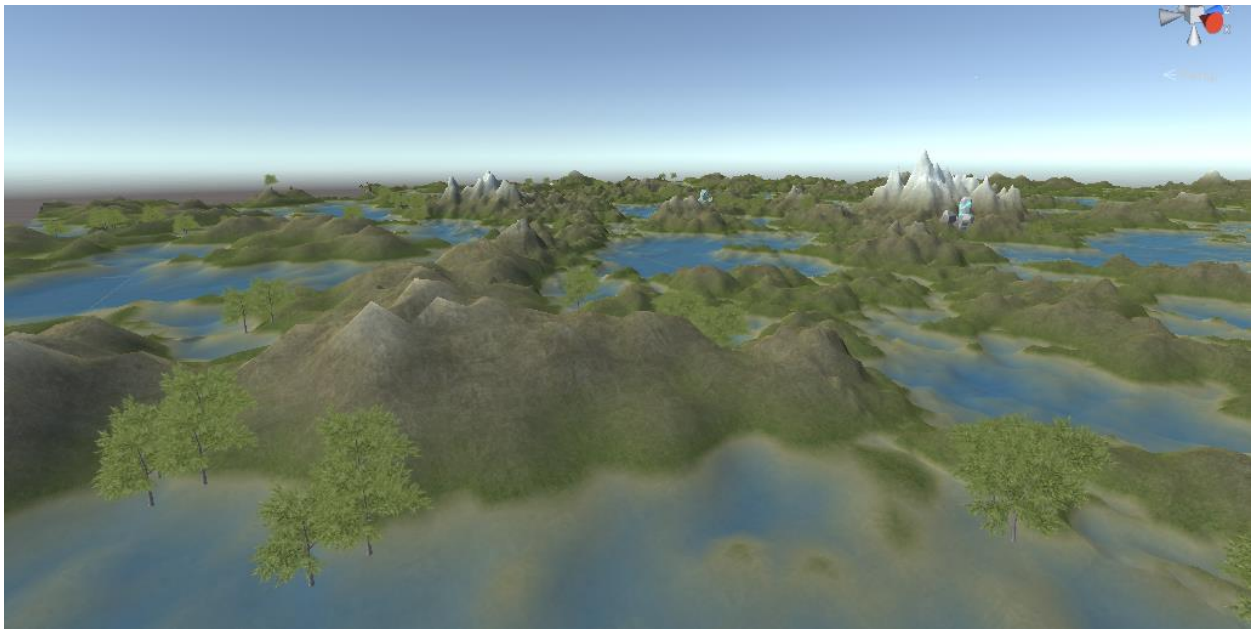

Рисунок 3.9 – Приклад згенерованого світу

Наповнення підземель також має певні особливості та частково використовує зазделегіть підготовлені елементи з позиціями, як раз у випадку використання освітлення даний підхід важливий у створенні атмосфери, бо таким чином ми можемо контролювати ситуацію для гравця, тому вхідними даними для цього є лише ліміт у кількості, драбини, певні старі атрибуті притаманні підземеллям також підготовленні і мають декілька наборів до генерування. Однак даний підхід підійде лише у випадку неживих об'єктів, а оскільки планувалося додавання ворогів, було вирішено додати тригерну генерацію, яка відповідно до встановленої складності та взаємодії гравця з кімнатою буде створювати монстрів, так само відбувалось додавання пасток, кожна з них отримує персональну криву с частотою активації та режимом роботи, а позиція обирається згідно до можливих позицій.

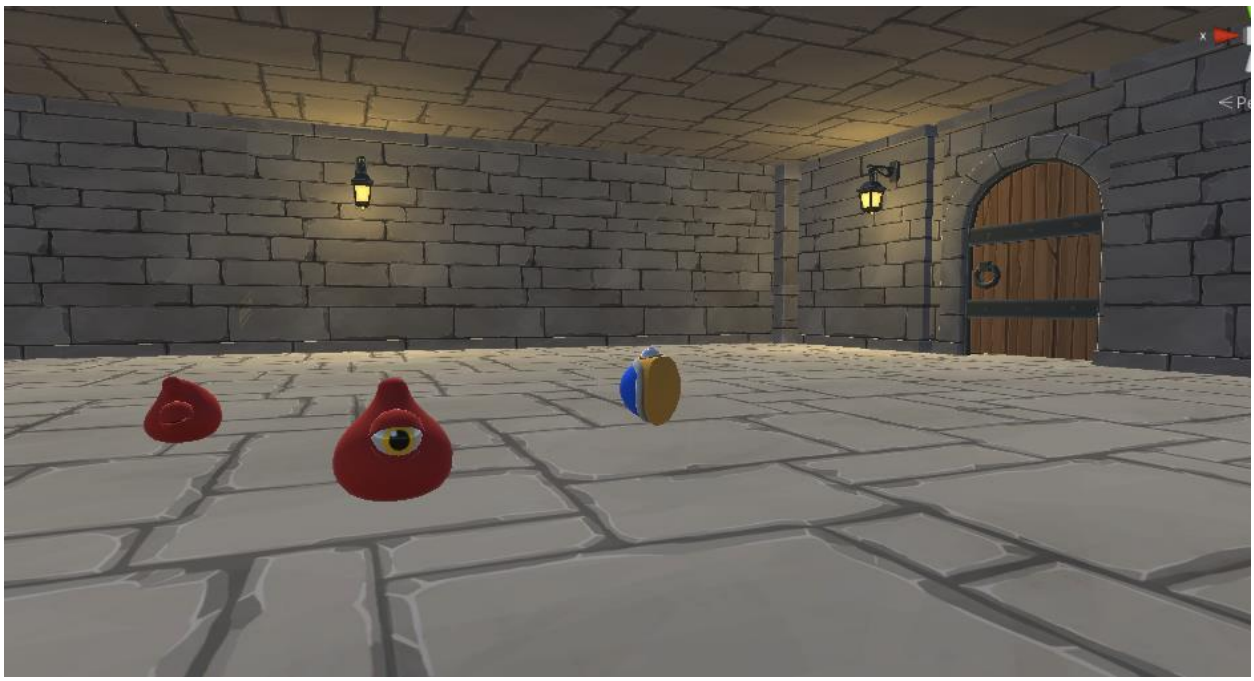

Рисунок 3.10 – Приклад згенерованих ворогів

Оскільки монстри дуже схожі, їм було надано також різні випадкові поведінки, які обираються в момент створення з зазделегіть підготовленної логіки.

Подібно простим монстрам так само генерується один з можливих фінальних босів.

4 ОПИС ПРОГРАМИ

4.1 Основні об’єкти

Як алгоритми процедурної генерації, і представлення лише на рівні програми містяться у спеціальних класах Dung та Map, де відповідно у кожного відбувається генерування, які успадковуються від базового класу у середвоищи Unity - MonoBehaviour. MonoBehaviour – це базовий клас, від якого успадковуються усі скрипти. При використанні JavaScript кожен скрипт автоматично успадковує MonoBehaviour. Якщо ви використовуєте C# або Boo, ви повинні успадкувати від MonoBehaviour.

Ігрове поле - це згенерований в результаті алгоритму рівень, що складається зі створених ландшафтів та кімнат у списку CurrentRooms. Всі зміни у функціоналі створених кімнат та зон здійснюються відповідно до дій користувача.

Кімната - це готовий префаб з компонентом Кімната, що відображається у вигляді набору вокселів у поєднанні з графічним дизайном.

Елементи кімнати - містить додаткові функції та елементи для кожної кімнати, такі як генератори монстрів та скрипти з можливістю знищувати певні випадкові елементи для додавання унікальності та унікальності.

4.2 Аналіз генерації підземелля

Те, як генеруються кімнати, дуже важливе при створенні ігрового поля, головною причиною є те, що в результаті виконання створюється кімната зі своїм особистим характером, і всередині неї генерується контент.

Лістинг 4.1 – Генерації кімнат

```
private void Generation()
```

```
{
    var rooms = new List<Room>();
    var doorViews = new List<DoorView>();

    var firstPoint = Random.Range(0,GetDungeonSetCount() - 1);
    var firstRoom =
Instantiate(data.sets[dungeonSet].spawns[firstPoint].gameObject)
;
    for (var i = 0; i < rooms.Count; i++) {
        for (var j = 0; j < rooms[i].doors.Count; j++) {
            if (rooms[i].doorViews[j].door != null)
               {
     continue;
                var door =
(Instantiate(data.sets[dungeonSet].doors[0].gameObject)).GetComp
onent<DoorView>();
                doorViews.Add(d);

     var doorView = rooms[i].doors[j];
     doorView.door = door;
     doorView.sharedDoor.door = door;
     var currentTranform = rooms[i].doors[j].transform;
     var doorTransform = d.gameObject.transform;
                doorTransform.position =
urrentTranform.position;
                doorTransform.rotation =
currentTranform.rotation;
                doorTransform.parent =
this.gameObject.transform;
            }
        }
    }
}
```

Згідно до коду в кожній кімнаті існує набір дверей, де відповідно додається компонент для аналізу та з’єднання кімнат.

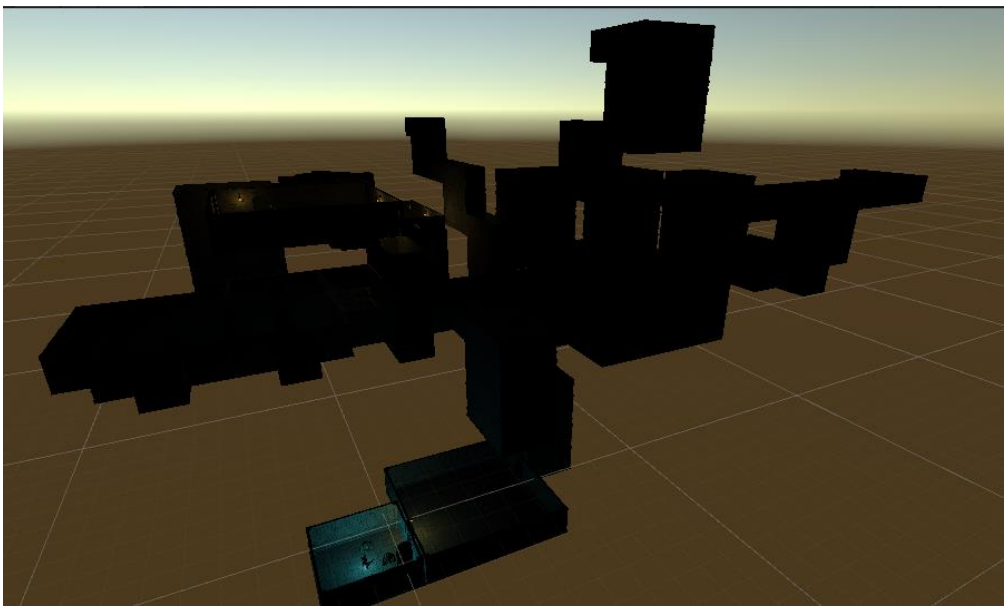

Рисунок 4.1 – Результат створення підземелля

Надалі розглянемо метод, що відповідає за підключення можливих кімнат, залежно від стану кімнатного конектора. Таким чином, якщо до кімнати додано 3 можливі варіанти підключення і є суміжні кімнати, позицій, що перекриваються, не буде. Створюється новий сусід, що ніяк не перехрещується з існуючим.

Лістинг 4.2 – Під’єднання кімнат

```
public DoorSettings Conect (Room last, Room current)
{
    var lastRoomDoor = last.GetRandomDoor(random);
    var newRoomDoor = current.FirtDoor(true);
    var roomTransform = newRoom.transform;
    var lastTransform = lastRoomDoor.transform;
    roomTransform.rotation =
GetAngle(lastTransform,roomTransForm);
    var resultPosition = lastTransform.position -
roomTransform.position;
    roomTransform.position += resultPosition;
    newRoomDoor.SetBounds(); return lastRoomDoor;
}
```

Важливо звернути увагу, як створюється нова кімната – її можлива ротація. Це може призвести до створення непрохідних та ідентичних

елементів.

Лістинг 4.3 – Генерація нової кімнати

```
private void RoomGenerator()
{
...

for (var j = 0; j < openSet.Count; j++) {
    for (var k = 0; k < openSet[j].doors.Count; k++) {
        if(!IsDoorOpen() || !IsDoorValid(door)) continue;
CalculateValideAngle();
        var resultDirection = new Vector3();
        switch(rotation)
{
        case 270:
        case 180:
        case 90:
        case 0:
        resultDirection = new Vector3(1f, 0f, 0f);
          break;
        default:
        resultDirection = Vector3.zero;
           break;
 }
        var gameObj =
GameObject.CreatePrimitive(PrimitiveType.Sphere);

     var objTransform = gameObj.transform;
objTransform.position =
openSet[j].doors[k].voxelOwner.transform.position +
(resultDirection * v.voxelScale);

        doorVoxelsTest.Add(g);
        overlap = GetOverlap() }}
```

Як можна примітити, що присутнє додавання тестового вокселю - це зроблено для додаткової перевірки, аби не було дверей пасток, згідно з поставленою умови, але може легко бути модифіковано.

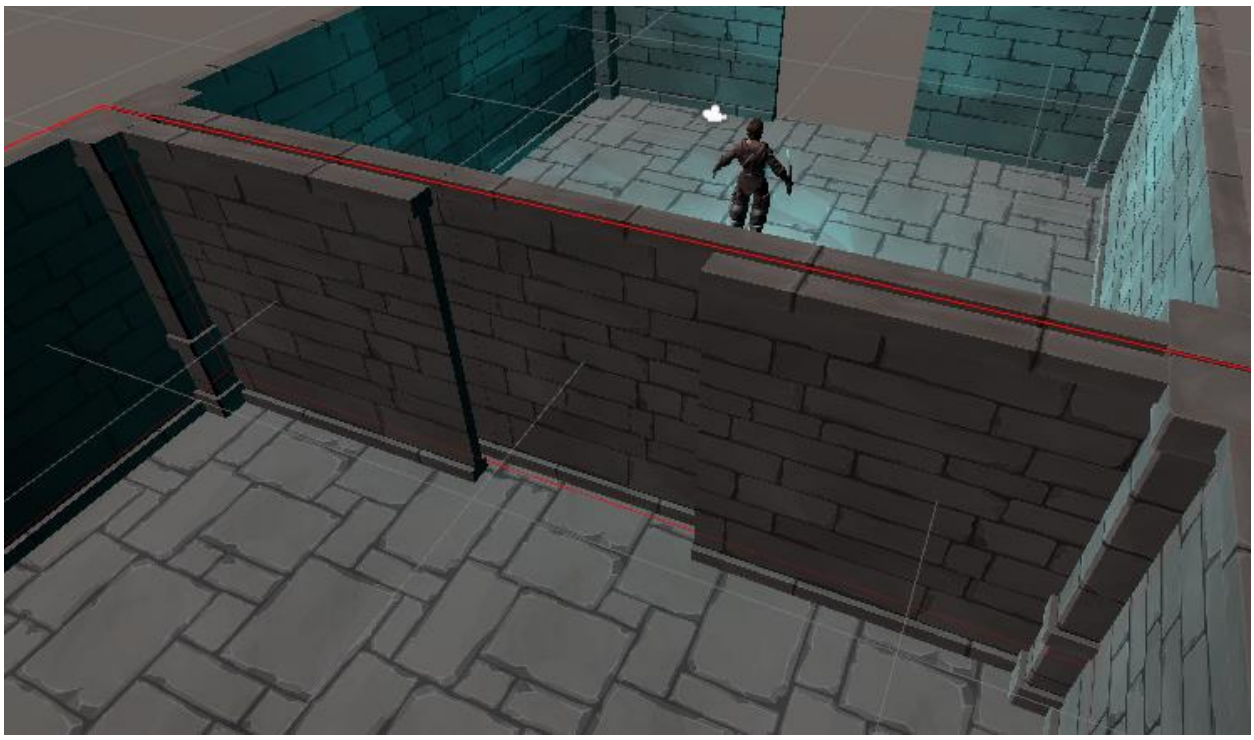

Рисунок 4.2 – Приклад генерації без повороту кімнат

4.3 Аналіз генерація ландшафту

При створенні ландшафту одним з найважливіших є сам метод генерації. Це пов'язано з тим, що в результаті його виконання створюються координати, які використовуються в матеріалі для результуючого графічного відображення.

Лістинг 4.4 – Генерації карти висот

```
private HeightMap Generate((int,int)size, HeightMapSettings
settings, Vector2 centre)
     {
          var noiseMap = Noise.Generate (size.Item1, size.Item2,
settings.Data, Centre);

          var animCurve = new AnimationCurve
settings.heightCurve.keys);

          var min = float.MaxValue;
          var max = float.MinValue;

          for (var x = 0; x < width; x++)
```

```
          {
               for (var y = 0; y < height; j++)
               {
                    var currentValue = values [x, j];
                    currentValue *= animCurve.Evaluate (value) *
settings.heightMultiplier;

                    if (currentValue > maxValue)
                         max = values [x, y];

                    if (currentValue < minValue)
                         min = values [x, y];
               }
          }

          return new HeightMap (values, minValue, maxValue);
     }
```

Давайте ближче познайомимося із класом. Цей клас використовується для створення нових ландшафтних об'єктів шляхом надання певного матеріалу та згенерованих координат висоти.

Лістинг 4.5 – Генерації чанку

```
     public Chunk(int number, Vector2 coord, LODInfo[] details,
int colliderIndex, Transform parent, Transform viewer, Material
material) {
          …

          resultObject = new GameObject("Terrain Chunk Number "
+ number);
          renderer = resultObject.AddComponent<MeshRenderer>();
          filter = resultObject.AddComponent<MeshFilter>();
          collider =
resultObject.AddComponent<MeshCollider>();
          renderer.material = material;

var transform = resultObject.transform;
          transform.position = new
          Vector3(position.x,0,position.y);
          transform.parent = parent;
          SetVisible(false);
```

Додатково до об’єкту мешу був доданий компонент Collider, саме використовуючи його надалі відбувається генерація наповнення світу, як

приклад дерева.

Рисунок 4.3 – Приклад генерації без додавання кольорів на матеріал

5 ТЕСТУВАННЯ ТА СИСТЕМНІ ВИМОГИ

5.1. Тестування

Нижче наведено приклад віртуального простору, згенерованого цим алгоритмом кваліфікаційної роботи. Параметри генерації змінили лише максимально допустиму кількість кімнат.

Рисунок 5.1 – Результат генерації з героєм

Згідно до очікувань, жодних проблеми у відображені, або генерації відсутні. Завантаження відбулося дуже щвидко, без присутніх проблем та злагоджено. Спробуємо перейти до сусідніх кімнат та перевірити генерацію їх наповнення.

Рисунок 5.2 – Перевірка на генерацію контенту в кімнатах

Відповідно до модулю додаткової генерації були утворені монстри, кількість їх відповідає встановленим значенням, логіка дій стабільна та без помилок, усе працює за умовою в завданні.

Спробуємо відкрити сусідні двері та перевірити кімнату на ідентичність.

Рисунок 5.3 – Перевірка сусідньої кімнати

Як невеликий висновок, можна сказати що наповнення динамічно створюється та змінюється відповідно до встановленої умови.

5.2. Системні вимоги

У кожного продукту повинні бути відповідні умови для запуску, оскільки продемонстрована реалізація додатку, була виконана у середовище Unity то прив’язка до платформи запуску відсутня, але потрібно мати 175 МБ вільного місця на жорстоку диску, а також рекомендовано мати вбудовану відеокарту.

5.3. Інструкція користувача

Ви повинні встановити програму. Щоб розпочати роботу, потрібно натиснути кнопку Start. На початку гравець знаходиться у вихідній позиції з можливістю переміщення, опиняючись у підземеллі при взаємодії з порталами, відкриваючи будь-які двері для дослідження створеного лабіринту

ВИСНОВКИ

В рамках даної кваліфікаційної роботи були розглянуті проблеми при створенні віртуальних просторів з точок зору як ручного підходу, так і процедурного, коли структури даних, що задають цей простір, генеруються за допомогою програмних алгоритмів.

Ця робота присвячена алгоритмам процедурної генерації рівнів, деякі з яких використовують тайлові уявлення рівнів на основі аналізу трьох існуючих алгоритмів, в результаті було створено новий алгоритм, саме якому і була присвячена основна частина роботи.

Тестування програми показало, що цей додаток має недоліки, які будуть виправлені в майбутньому. Ви можете зіткнутися з такими проблемами, як відкриття дверей або неправильна поява об'єктів. Рекомендується запобігти цьому недоліку в майбутніх версіях програми.

Хоч і можна практично застосовувати цю програму в побуті, не всім комфортно грати в цей продукт, але у професійній сфері він може скоротити час розробки світу майже на третину, тому потенціал у нього великий. Для більш комфортного використання програму необхідно доопрацювати у деяких аспектах. Одним із них є можливість редагувати генерацію підземель та комбінувати згенеровані ландшафти.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Проблеми інформатизації. Тези доповідей десятої міжнародної науково-технічної конференції Том 2: секція 4 – 2022. – 113 с.

2.Ріхтер Д. Програмування на платформі Microsoft .NET Framework 4.0 на мові C#/ Д. Ріхтер. – Санкт-Петербург: «Пітер», 2012. – 1136 с.

3.Шілдт Г. C# 4.0. Повне керівництво / Г. Шілдт. – Київ: Видавничий дім «Вільямс»– Київ, 2011 – 1613 с.

4.CLR via c#- Джеффри Рихтер

5.Стілмен Э. Вивчаєм C#/ Э. Стілмен. – Санкт-Петербург: «Пітер», 2012. – 1454 с.

6.MSDN Developer Network [Электронний ресурс] - Режим доступу: www/ URL: https://msdn.microsoft.com/ – 04.12.2022 р. – Загол. з екрану.

7. Map representations [Електроний ресурс] – Режим доступу: www/ URL: http://theory.stanford.edu/~amitp/GameProgramming/MapRepresentations.html – 04.12.2022 р. – Загол. з екрану.

8. Unity Documentation [Електроний ресурс] – Режим доступу: www/ URL: https://docs.unity3d.com/Manual/index.html – 04.12.2022 р. – Загол. з екрану.

9.GenerateRandomCaveLevelsUsingCellularAutomata [Електроний ресурс] – Режим доступу: www/ URL: http://gamedevelopment.tutsplus.com/ tutorials/generate-random-cave-levels-using-cellular-automata--gamedev-9664 – 04.12.2022 р. – Загол. з екрану.

10. Алгоритм «diamond-square» для построения фрактальных ландшафтов [Електроний ресурс] – Режим доступу: www/ URL: https://habr.com/ru/post/
111538/ – 04.12.2022 р. – Загол. з екрану.

11. Генерация трехмерных ландшафтов. [Електроний ресурс] – Режим

доступу: www/ URL: https://www.ixbt.com/video/3dterrains-generation.shtml – 04.12.2022 р. – Загол. з екрану.

12.Generate random cave levels using cellular automata [Електроний ресурс] – Режим доступу: www/ URL: https://gamedevelopment.tutsplus.com/ tutorials/generate-random-cave-levels-using-cellular-automata--gamedev-9664 – 04.12.2022 р. – Загол. з екрану.

13. Процедурная генерация уровней [Електроний ресурс] – Режим доступу: www/ URL: https://habr.com/post/418685/ – 04.12.2022 р. – Загол. з екрану.

14. Procedural Map Generation [Електроний ресурс] – Режим доступу: www/ URL: https://www.gridsagegames.com/blog/2014/06/procedural-map-generation – 04.12.2022 р. – Загол. з екрану.

15.Benes,B.,Androgbatch,R. Layered data represen-tation for visual simulation of terrain erosion. InProc. of the Spring Conf. on Comp. Graphics, 2001. - 80–85c.

16. Технологии компьютерных игр [Электронний ресурс]. – Режим доступу: www/ URL: http://mirznanii.com/a/115426/tekhnologii-kompyuternykh-igr/ – 04.12.2022 р. – Загол. з екрану.

17. 7 примеров использования процедурной генерации в играх, о которых полезно знать всем разработчикам [Электронний ресурс]. – Режим доступу: www/ URL: https://habrahabr.ru/company/ua-hosting/blog/275195/ – 04.12.2022 р. – Загол. з екрану.

18. How to use BSP trees to generate game maps [Электронний ресурс]. – Режим доступу: www/ URL: https://gamedevelopment.tutsplus.com/tutorials /how-to-use-bsp-trees-to-generate-game-maps--gamedev-12268 – 04.12.2022 р. – Загол. з екрану.

19.Процедурная генерация планетарных карт [Электронний ресурс]. – Режим доступу: www/ URL: https://habrahabr.ru/post/313420 – 04.12.2022 р. – Загол. з екрану.

20.Генерация ландшафта как в MineCraft [Электронний ресурс]. –

Режим доступу: www/ URL:https://habrahabr.ru/post/128368 – 04.12.2022 р. – Загол. з екрану.

21. David M.Mount. Computer Graphics. Lecture notes. Dept. of Computer Science. University of Maryland, 1997.

22. Fortune’s algorithm and implementation[Електроний ресурс] – https://blog.ivank.net/fortunes-algorithm-and-implementation.html– 04.12.2022 р. – Загол. з екрану.

23. Procedural Generation in Game Design. Tanya Short, Tarn Adams, 2017.

24. Procedural Content Generation in Games. Noor Shaker , Julian Togelius , Mark J. Nelson, 2016.