НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Завідувач кафедри
_____ Олександр КОВАЛЬ
«___»_____2021 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Геометричне моделювання процесів та систем»

спеціальності 122 «Комп'ютерні науки»

на тему: «Застосування фрактальної геометрії при побудові карти відеоігри»

Виконала:
студентка IV курсу, групи ТР-72
Паламар Інна Олегівна _____

Керівник:
Доцент, к.т.н.,
Залевська Ольга Валеріївна _____

Консультант:
_____

Рецензент:
Доцент,к.т.н.
Можаровський В.М. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ — 2021 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 122 Комп’ютерні науки

Освітня програма: Геометричне моделювання процесів та систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр КОВАЛЬ
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту
Паламар Інні Олегівни
(прізвище, ім’я, по батькові)

1. Тема роботи _Фрактали у відеоіграх_____

керівник роботи_____ Залевська Ольга Валеріївна , к.т.н.
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”24” травня 2021р. № **267-с**

2. Строк подання студентом роботи _____08.06.2021р._____
3. Вихідні дані до роботи _мова програмування C#, ігровий рущій Unity , середовище розробки Visual Studio Code

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) розробити програмне забезпечення для імітації відеогри з використанням фрактальної графіки та фрактальний зв’язок взаємодії гравця з навколишнім ігровим середовищем.

5. Перелік ілюстративного матеріалу

Вступ , Фрактали, Розробка гри, Фрактали у відеоіграх, Фрактальні лабіринти, Рекурсивний метод, Килим Серпінського, Приклади реалізації, Недоліки та переваги використання фракталів, Висновки.

6. Консультанти розділів роботи

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

7. Дата видачі завдання "_10_"_____жовтня_____ 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	09.10.2020	
2.	Вивчення та аналіз задачі	10.10.2020-16.12.2020	
3.	Розробка архітектури та загальної структури системи	12.04.2021-25.04.2021	
4.	Розробка структур окремих підсистем	26.04.2021-02.05.2021	
5.	Програмна реалізація системи	03.05.2021-09.05.2021	
6.	Оформлення пояснювальної записки	12.03.2021-04.06.2021	
7.	Захист програмного продукту	11.05.2021	
8.	Передзахист	25.05.2021	
9.	Захист	14.06.2021	

Студент _____ _____Паламар І.О._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Залевська О.В._____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

У роботі розглянуті теоретичні та практичні аспекти фрактальної графіки, використання фракталів для моделювання взаємовз’язку між персонажем та навколишнім середовищем. Реалізувати розглянуті алгоритми та методи на прикладі відеогри «Лабіринт».

Метою роботи є створення відео гри в основі якої лежить пошук коробок у випадково згенерованому фрактальному лабіринті.

Для досягнення мети розроблено схематичну класифікацію фрактальних лабіринтів, реалізовано алгоритми рекурсивного ділення та побудови килима Серпінського, розглянуті методи реалізовані за допомогою фрактальної графіки та протестовано застосунок.

Пояснювальна записка складається зі вступу, шести розділів, висновку, списку використаних джерел; містить 57 сторінки, 16 рисунки, 14 таблиці та 5 додатки. Список використаних джерел включає 22 бібліографічних найменувань.

Ключові слова: відео гра, фрактальні лабіринти, Unity, алгоритм рекурсивного ділення, килим Серпінського.

ABSTRACT

The paper considers the theoretical and practical aspects of fractal graphics, the use of fractals to model the relationship between the character and the environment. Implement the considered algorithms and methods on the example of the video game "Labyrinth".

The aim of the work is to create a video game based on the search for boxes in a randomly generated fractal maze.

To achieve this goal, a schematic classification of fractal labyrinths was developed, algorithms for recursive division and construction of the Serpinsky carpet were implemented, the considered methods were implemented using fractal graphics and the application was tested.

The explanatory note consists of an introduction, chapters, conclusion, list of used sources; contains 57 pages, 16 figures, 14 tables and 5 applications. The list of used sources includes 22 bibliographic items.

Key words: video game, fractal mazes, Unity, recursive division algorithm, Serpinsky carpet.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ВФЛ	—	Віртуальні фрактальні лабіринти
КС	—	Килим Серпінського
C#	—	C Sharp
ПК	—	Персональний комп’ютер

ВСТУП

З кожним днем відеоігри набувають все більшої популярності, зростає й попит на нові креативні рішення як і у ігровому процесі, так і у графіці. Велику роль у вирішені даної проблеми грають фрактали. Вони знайшли багато варіантів використання: від зображення дерев та об’єктів у 3D просторі, побудови ландшафтів карт до інтеграції їх у гемплей на біль глибшому рівні, як наприклад побудови основної концепції гри коли однакові невеликі частини гри є частинами більшої, але все ще подібною на них по концепції.

З цього постає необхідність у пошуку нових способів використання фрактальних алгоритмів для покращення вражень від процесу гри, як і у графічному, так і у концептуальному рівні.

У даній роботі буде розглянено не лише способи використання фракталів для побудови ландшафтів карт, а й для генерації складних лабіринтів, алгоритми процедурної генерації 3D лабіринт в відеогрі на основі кросплатформлений ігровий двигун Unity та можливість його використання у інших типах відеоігор.

1. ЗАДАЧА ВИКОРИСТАННЯ ФРАКТАЛІВ У ВІДЕОІГРАХ

Проаналізувати специфіку використання фракталів при створені відеоігор, розглянути варіанти процедурної генерації лабіринтів, розглянути існуючі програмні реалізації, а також вказати їх переваги та недоліки, розглянути засоби для створення відеоігор, проаналізовати їх переваги та недоліки, описати та розробити основні функції програмної реалізації.

Задачі, які потрібно розв’язати для досягнення мети:

- Провести аналіз існуючих способів використання фракталів при створені відеоігор.
- Проаналізувати існуючі алгоритми побудови лабіринтів.
- Проаналізувати існуючи засоби для створення відеоігор у 3Д просторі.
- Проаналізувати концепти ігрового процесу на базі використання фракталів.

2. МЕТОДИ ПОБУДОВИ ФРАКТАЛІВ ТА ЛАБІРИНТІВ

2.1 Основні відомості про побудову фракталів

Фрактали – однорідні, самоподібні об’єкти різьбляної форми. Вперше термін був введений Бенуа Мандельброт у 1975 році. Самі ж фрактали досліджувалися задовго до того, як отримали таку назву. Це зумовлено тиям, що вони оточують нас, являючісь невідьємленою частиною природи.

Об’єкти що володіють фрактальними властивостями:

В живій природі	В неживій природі
Корали	Межі географічних об’єктів
Морські зірки	Берегові лінії
Рослини та квіти	Гірські хребти
Плоди (ананас)	Сніжинки
Крони дерев	Блискавки
Кровоносна система	Кристали
Бронхи	Сталактити, сталагміти, геліктити

Таблиця 2.1.1 - Об’єкти що володіють фрактальними властивостями.

Широке розповсюдження фрактали знайшли у фізиці. Виділяють такі групи фізичних явищ: агрегація, випадкові блукання та дифузія, явища протікання та перколяція, динамічний хаос. Інші розділи фізики, використовують фрактальні підходи побудовані на цій групі. В свою чергу

фрактали широко використовуються для стискання інформації, опису процесів в природніх науках.

В математиці, приклади самоподібних структур з патологічнимивластивостями, відносно класичного аналізу, з'явилися лише наприкінці XIX століття. До них можна віднести наступні:

Множина Кантора	ніде не щільна незліченна досконала множина. Модифікувавши проце-дуру, можна також отримати ніде не щільну множину позитивної дов-жини
Трикутник Серпінського («скатертина») і килим Серпінського	налоги множини Кантора на площині
Губка Менгера	налог множини Кантора в триви-мірному просторі;
Приклади Веєрштрасса і ван дер Вардена	ніде не диференційованої непере-рвної функції
крива Коха	неперервна крива, що не перетинається, нескінченної довжини, яка не має дотичній ні в одній точці
крива Пеано	неперервна крива, що проходить через всі точки квадрата

Таблиця 2.1.2 - Самоподібні множини з незвичайними властивостями в математиці

З розвитком та поглибленням вивчання фракталів з'являлося все більше способів та методів їх генеруванняю До поширених можна віднести наступні:

Назва	Опис	Приклади
Ітераційні функції	будуються відповідно до фіксованого правила геоме-тричних заміщень.	Множина Кантора, ки-лим Серпінського, крива Пеано, крива Коха, крива дракона
Рекурентні співвідношення	Фрактали, що визначаються рекурентним відношенням в кожній точці простору	множина Мандельброта, палаючий корабель та фрактал Ляпунова.
Випадкові проце-си	Фрактали, що генеруються з використанням стохастичних, а не детермінованих процесів	фрактальні ландшафти, траєкторія Леві та броунівське дерево

Таблиця 2.1.3 – методи побудови фракталів.

2.2 Способи використання фракталів у відеоіграх

Фрактали самі по собі не лише красиві, але вони можуть бути і неймовірно корисними в комп'ютерній графіці. На основі вивчених фрак-тальних закономірностей, знайдених в природі, можливо створювати програми для їх імітації. Вивчаючи схеми розгалуження дерев, можна процедурно генерувати дерева програмним шляхом, а трохи змінюючи параметри цих фракталів, ви можете створити нескінченну кількість уні-кальних, реалістичних дерев.

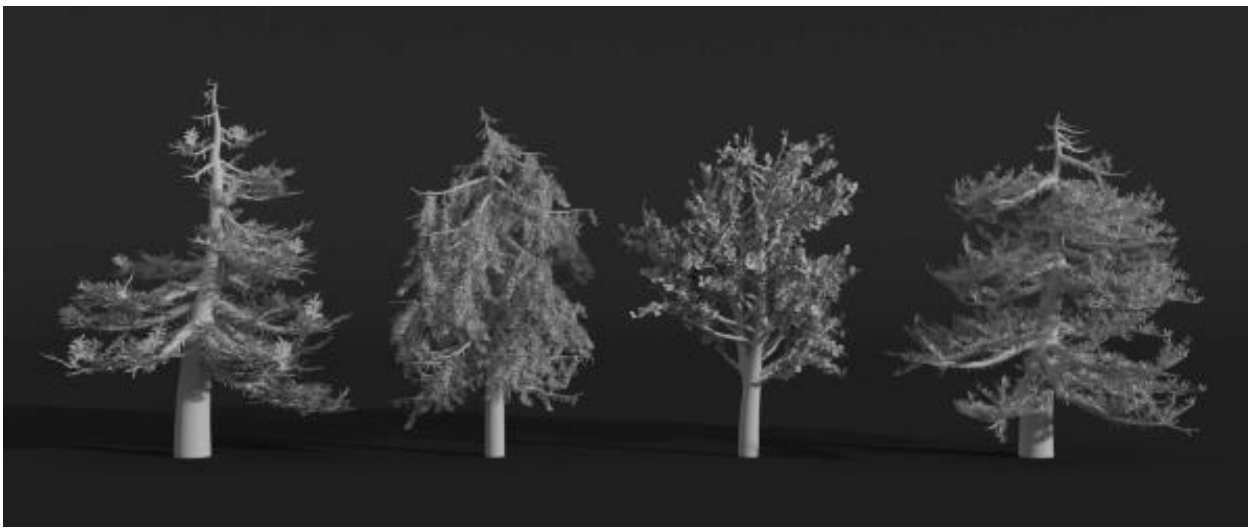


Рисунок 2.2.1 — Приклад побудови фрактальних дерев

Ці фрактали також можна застосовувати в більших масштабах, на-приклад, для створення ландшафтів. Багато реальних географічних об’єктів де-онструють фрактальну схожість: узбережжя, гори та річки. Таким же чином фрактали використовують для генерації місцевостей процедурно, без необхід-ності моделювати вручну.

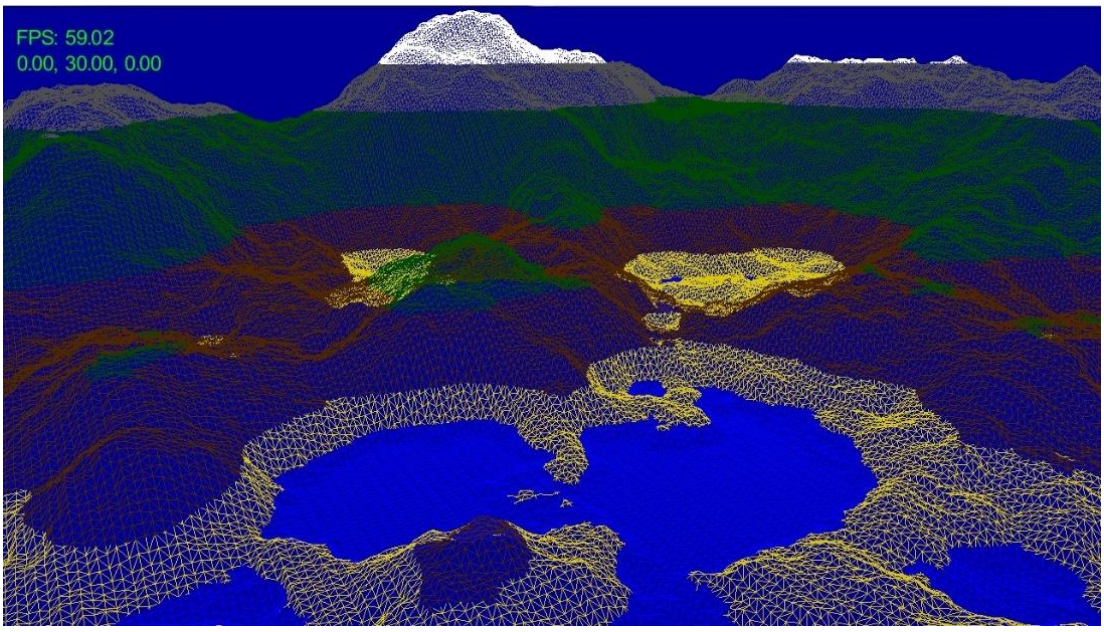


Рисунок 2.2.2 — Приклад фрактальної генерації ландшафту

Однак, хоча ці прийоми можуть бути дуже корисними не лише у питані графічного зображення середовища, «фрактальним дизайном гри». Подібно до

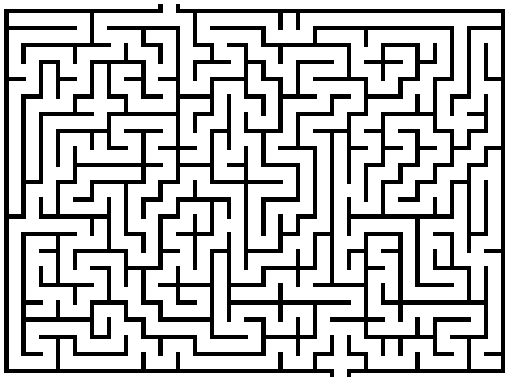
того, як фрактал виглядає схожим при збільшенні та зменшенні масштабу, деякі відеоігри роблять те саме - не на візуальному рівні, а на механічному.

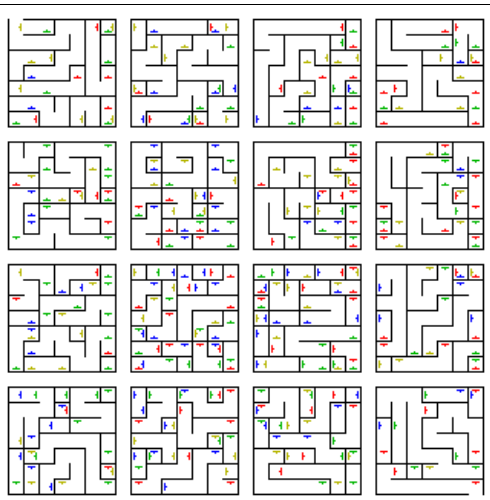
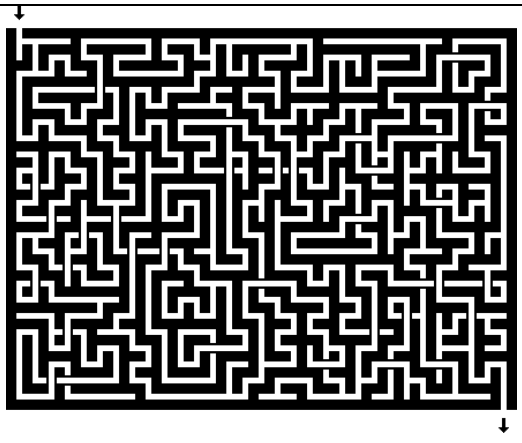
2.3 Лабіринти: класифікація, генерування, пошук рішень

Лабіринти в цілому (а значить, і алгоритми для їх створення) можна розбити по семи різних класифікацій: розмірності, гіперразмерності, топології, тесселяции, маршрутизації, текстурі і пріоритету. Лабіринт може використовувати по одному елементу з кожного класу в будь-якому поєднанні.

2.3.1 Класифыкацыя

Розмірність: характеризує кількість вимірів у просторі.

Назва	Приклад	Опис
Двомірні		Більшість лабіринтів, як паперових, так і реальних, мають цю раз-мірність, тобто ми завжди можемо відобразити план лабіринту на аркуші паперу і рухатися по ньому, не перетинаючи ніяких інших коридорів лабіринту.

Тривимірні		Тривимірний лабіринт має кілька рівнів; в ньому (по мірі, в ортогональної версії) проходи можуть крім чотирьох сторін світу опус-каються вниз і підніматися вгору. 3D-лабіринт часто візуалізують як масив з 2D-рівнів з позначками сходів «вгору» і «вниз».
Більш високі розмірності		Можна створювати чотиривимірні лабіринти і ще більш багатовимірні. Іноді їх візуалізують як 3D-лабіринти з «порталами», провідними крізь четвертий вимір, наприклад, портали в «минуле» і «майбутнє».
Переплетення		Лабіринти з переплетеннями - це, по суті двомірні (або, точніше 2,5-мірні) лабіринти, в яких, проте, проходи можуть накладатися один на одного. При відображенні зазвичай цілком очевидно, де знаходяться тупики та як один прохід знаходиться над іншим. Лабіринти з реального світу, в яких є мости, що з'єднують одну частину

		лабіринту з іншого, частково є переплетеннями.

Таблиця 2.3.1.1 – класифікація лабіринтів по розмірності

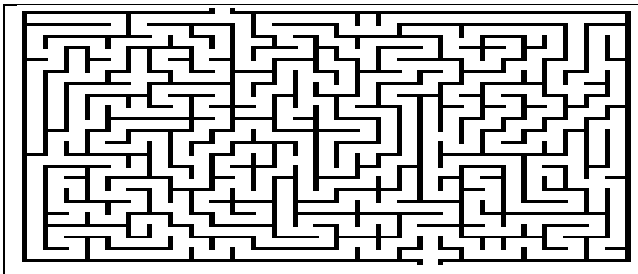
Гіперразмірність: відповідає розмірності об'єкта, що рухається через лабіринт, а не самого лабіринту.

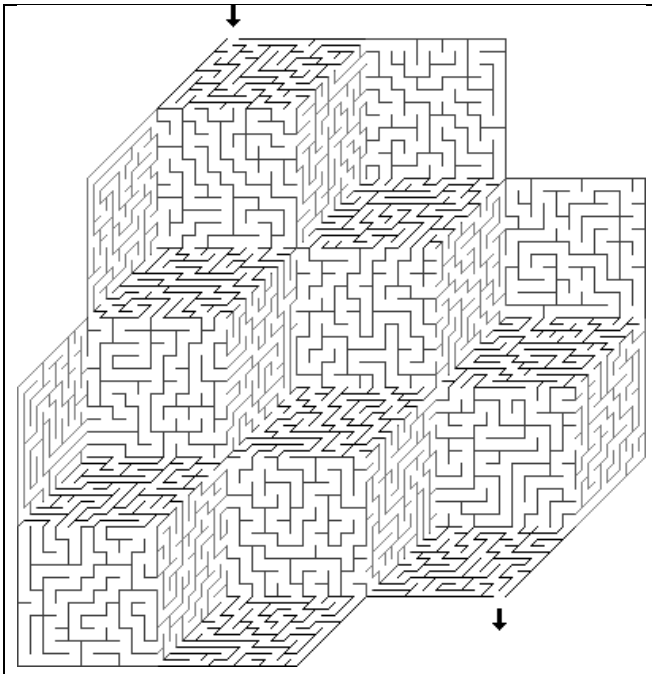
Назва	Опис
Не-гіперлабірінти	практично всі лабіринти, навіть створені у високій розмірності або мають особливі правила, зазвичай не-гіперлабірінтамі. У них ми працюємо з точкою або невеликим об'єктом, наприклад, кулькою або самим гравцем, які рухаються від точки до точки, а прокладений маршрут утворює лінію. У кожній точці існує легко прораховується кіль-кість варіантів вибору.
Гіперлабірінти	це лабіринти, в яких вирішується об'єкт є не просто точкою. Стандартний гіперлабірінт (або гіперлабірінт першо-го порядку) складається з лінії, яка при переміщенні по шляху утворює поверхню. Гіперлабірінт може існувати тільки в 3D або середовищі з більшою розмірністю, і входом в ги-перлабірінт часто є не точка, а лінія. Гіперлабірінт фундаментально відрізняється, тому що враховувати і одночасним працювати з декількома частинами уздовж лінії, а в будь-який момент часу існує практично нескінченну кількість станів і варіантів того, що можна зробити з лінією. Лінія рішення

	нескінченна, або її кінцеві точки знаходяться за межами гіперлабірінта, щоб уникнути стиснення лінії в точку, адже в цьому випадку можна вважати це не-гіперлабірінтом.
Гіпер-гіперлабірінт	можуть бути довільно високої розмірності. Гіпер-гіперлабірінт (або гіперлабірінт другого порядку) знову збільшує розмірність вирішуємого об'єкта. Тут вирішується об'єкт - це площина, яка при переміщенні по шляху утворює об'ємну фігуру. Гіпер-гіперлабірінт може існувати тільки в 4D або середовищі більшої розмірності.

Таблиця 2.3.1.2 – класифікація лабіринтів по гцперрозмірності

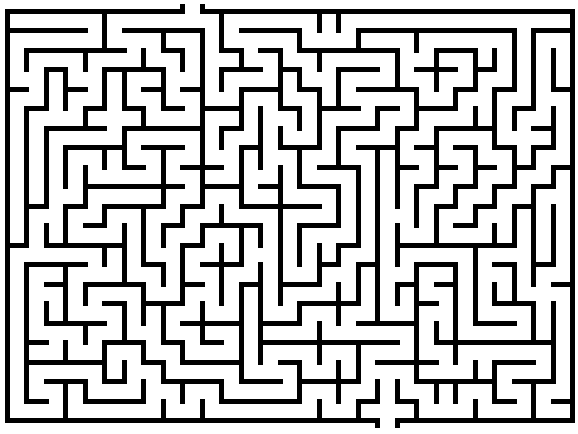
Топологія: клас топології описує геометрію простору лабіринту, в якому той існує як ціле.

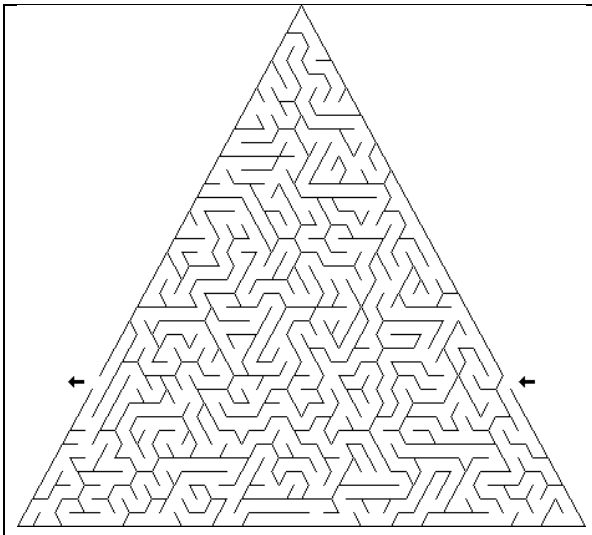
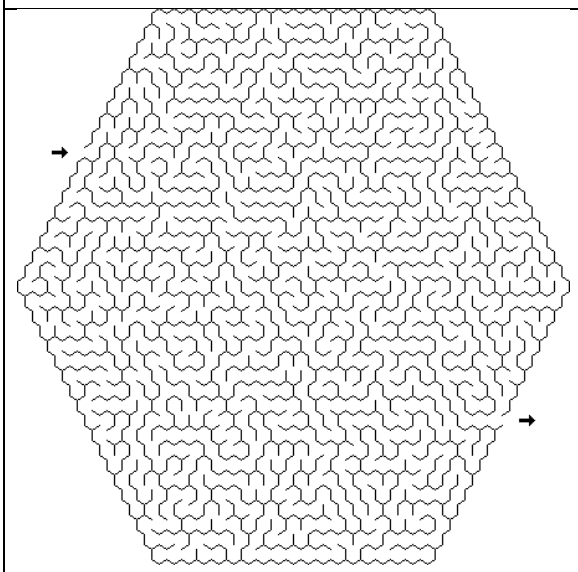
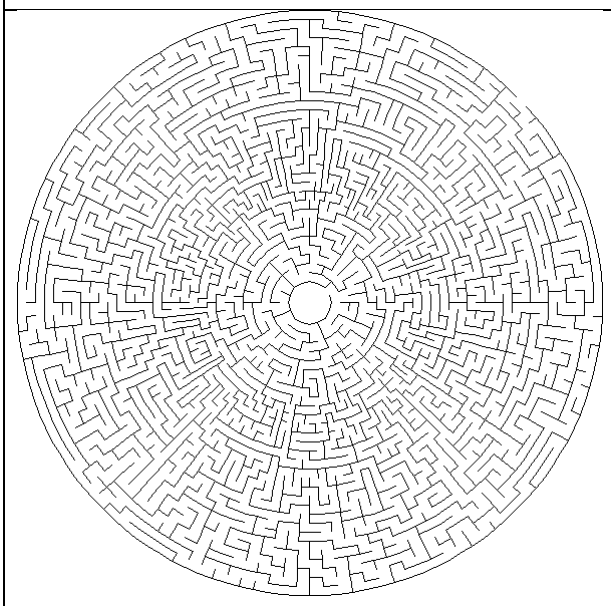
	Звичайний: це стандартний лабіринт в евклідовому просторі.

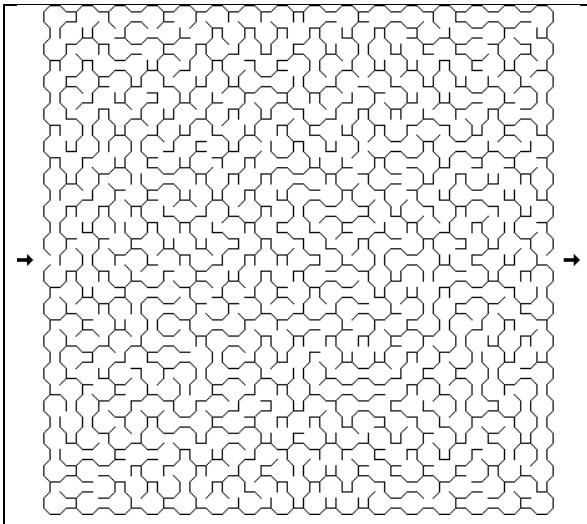
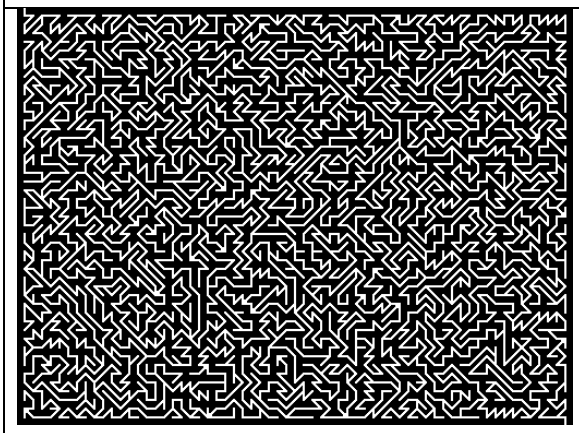
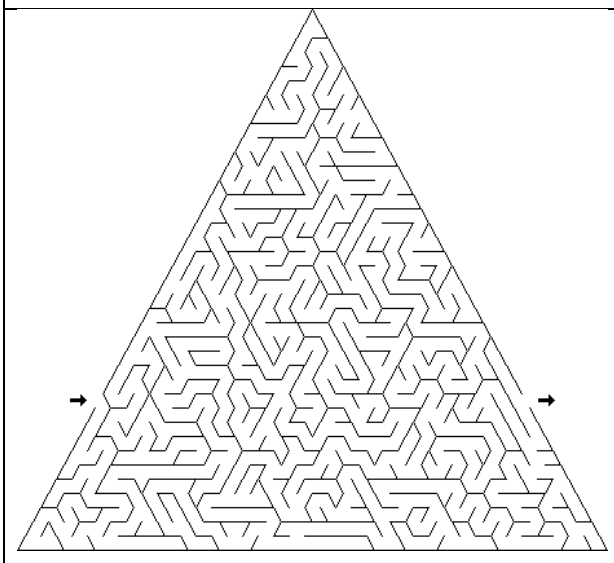
	Planair: термін «planair» описує будь-лабіринт з незвичайний-ний топологією. Зазвичай це означає, що краю лабіринту з-з'єднані цікавим чином. Приклади: лабіринти на поверхні куба, лабіринти на поверхні стрічки Мебіуса і лабіринти, еквівалентні знаходяться на торі, де попарно з’єднані ліва і права, верхня і нижня сторони.

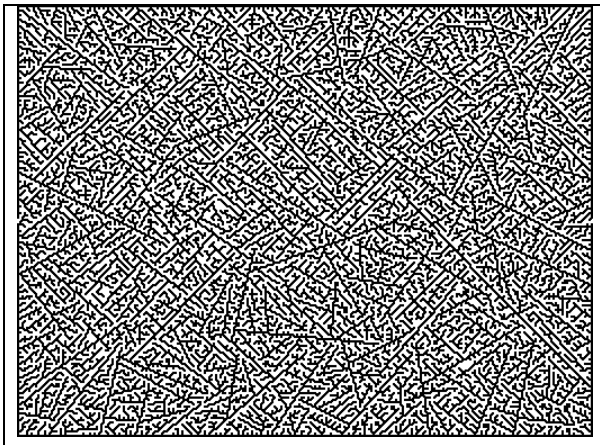
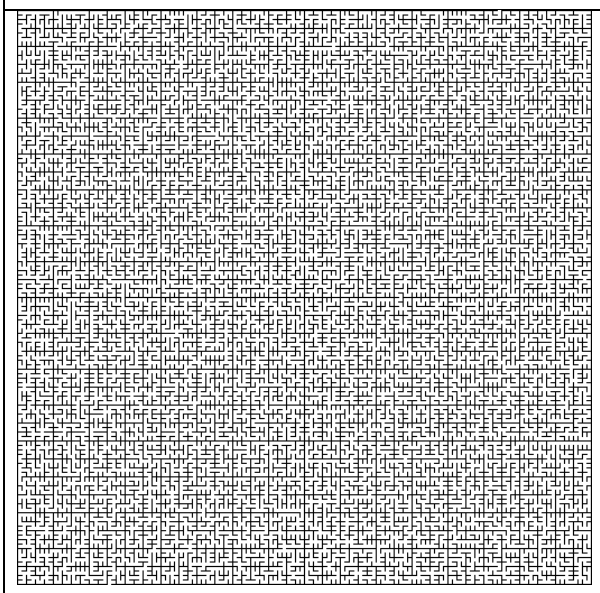
Таблиця 2.3.1.3 – класифікація лабіринтів по топології

Тесселяція: класифікація геометрії окремих осередків, з яких складається лабіринт.

Вид	Опис
	Ортогональний: це стандартна прямокутна сітка, в котрій осередки мають проходи, що перетинаються під прямими кутами. В контексті тесселяции їх також можна назвати гамма-лабіринтами.

	Дельта: дельта-лабіринти складаються із сполучених трикутників, і у кожного осередку може бути до трьох з'єднаних з нею проходів.
	Сигма: сигма-лабіринти складені із сполучених шести-кутників; у кожного осередку може бути до шести проходів.
	Тета: тета-лабіринти складаються з концентричних кіл проходів, в яких початок або кінець знаходиться в центрі, а інший - на зовнішньому краї. Осередки зазвичай мають чотири візмужніх з'єднання проходів, але їх може бути і більше блацію більшій кількості осередків в зовнішніх кільцях проходів.

	Іпсилон: іпсилон-лабіринти складаються із сполучених восьми-кутників або квадратів, в них кожна клітинка може мати до восьми або чотирьох проходів.
	Дзета: дзета-лабіринт розташований на прямокутній сітці, тільки на додаток до горизонтальних і вертикальних проходам допускаються діагональні проходи під кутом 45 градусів.
	Омега: термін «омега» відноситься практично до будь-якого лабіринту з постійною неортогональної тесселяції. Дельта-, сиг-ма, тета-і іпсилон-лабіринти відносяться до цього типу, як і багато інших схеми, які можна придумати, наприклад, лабірінт, що складається з пар прямокутних трикутників.

	Crack: crack-лабіринт - це аморфний лабіринт без постійногоної тесселяции, в якому стіни і проходи розташовані під випадковими кутами.
	Фрактальний: фрактальний лабіринт - це лабіринт, склад-ний з менших лабіринтів. Фрактальний лабіринт з вложені осередків - це лабіринт, в кожному осередку якого розбещений інші лабіринти, і цей процес може повторюватися кілька разів. Нескінченно рекурсивний фрактальний лабіринт - це справжній фрактал, в якому вміст лабіринту ко-бенкетує себе, створюючи по суті нескінченно великий лабіринт.

Таблиця 2.3.1.4 – класифікація лабіринтів по треселяції

Маршрутизація: описує типи проходів по лабіринту в межах його геометрії описаних вище.

Назва	Опис
Ідеальний	«ідеальним» називають лабіринт без петель або замкнутих ланцюгів і без недосяжних областей.
Плетений (Braid)	«плетений» означає, що в лабіринті немає тупиків.

Одномаршрутні (Unicursal)	під одномаршрутні подрузімевается лабіринт без розвилок
Розріджений	розріджене лабіринт Не прокладати проходи через кожну клітинку, тобто деякі з них не створюються.
Частково плетений	частково плетений лабіринт - це змішаний лабіринт, в якому є і петлі, і тупики

Таблиця 2.3.1.5 – класифікація лабіринтів по маршрутизації

Текстура: по суті своїй класифікує лабіринти по стилю проходів при різній маршрутизації і геометрії. Це не просто параметри, які можна вимикати або виключати при бажані.

Пріоритет: описує процеси побудови лабіринта, які можна розділити на два основні типи: додавання стіни і вирізання проходу. Іншими словами класифікація алгоритмів побудови лабіринтів за порядком дій, а не самих лабіринтів.

Назва	Опис
Додавання стін	алгоритми, для яких пріоритетом є стіни, починають з порожньою області (або зовнішнього кордону), в процесі роботи додаючи стіни. У реальному світі справжні лабіринти, що складаються з огорож, перекриттів або де-ревучи стін, безумовно є що додають стіни.

Вирізання проходів	алгоритми, пріоритетом яких є проходи, починають зі суцільного блоку і в процесі ра-боти вирізують в ньому проходи. У реальному світі такими лабирінту є тунелі шахт або лабіринти всередині труб.
Шаблон	зрозуміло, лабіринти можуть одночасно і Вире-мовити проходи, і додавати стіни; деякі комп'ютерні алгоритми так і роблять. Шаблоном лабіринту називають графічне зображення, яка не є лабіринтом, яке за най-меншу кількість кроків перетворюється на справжній лабіринт, але все одно зберігає текстуру вихідного графічного шаблона. Складні стилі лабіринтів, наприклад, пересічні спіралі, простіше реалізувати в комп'ютері як шаблони, а не намагатися створити правильний лабіринт, в той же час зберігаючи його стиль.

Таблиця 2.3.1.6 – класифікація лабіринтів по пріоритету

2.3.2 Алгоритми створення лабіринтів

Загальні типи алгоритми для побудови лабіринтів, представлені в таблиці 2.3.2:

Ідеальний	для створення стандартного ідеального лабіринту зазвичай необхідно «вирощувати» його, забезпечивши відсутність петель і ізольованих областей.
Плетений	для створення лабіринту без тупиків по суті потрібно до-додавали в лабіринт сегменти стін випадковим чином, але робити так, щоб кожен новий додається сегмент не приводив до створення тупика.
Одномаршрутні	один із способів створення випадкового одно-маршрутного лабіринту - створити ідеальний лабіринт, закрити вихід, щоб залишився тільки один вхід, а потім додавати стіни, що розділяють кожен прохід на дві частини.
Розріджений	розріджені лабіринти створюються рішенням не вирощувати лабіринт в областях, які будуть порушувати правило розрідженості.
3D	тривимірні лабіринти і лабіринти більшої розмірності можна створювати точно так же, як стандартний двомірний ідеальний лабіринт, тільки з кожного осередку можна випадковим чином рухатися в шість, а не в чотири ортогональні осередки. Через даткові розмірності в цих лабіринтах зазвичай використовується вирізання проходів.
Переплетених	переплетені лабіринти по суті створюються як ідеальні лабіринти з вирізанням проходів, тільки при вирізу-ванні проходів ми не завжди обмежені існуючим проходом, тому що у нас є можливість пройти під ним і все одно з-зберігати ідеальність лабіринту.
Crack: Crack	лабіринти по суті створюються як ідеальні лабіринти з додаванням стін, тільки в них немає виразної

	тесселяции, за винятком випадкового розташування пікселів.
Омега	для лабіринтів в стилі «омега» необхідно поставити якусь сітку, спосіб з'єднання осередків один з одним і прив'язку до екрану вершин, що оточують кожну клітинку.
Гіперлабірінт	це 3D-середовище, схожа на обернену версію стандартного тривимірного НЕ-гіперлабірінта, в котрому блоки стають відкритими просторами, і навпаки. Хоча стандартний 3D-лабіринт складається з дерева проходів через площю, гіперлабірінт складається з дерева смуг або ліан, що проходять по відкритій площі.
Planair	лабіринти з незвичайною топологією зазвичай створюється як масив з одного або декількох лабіринтів або частин лабіринтів, в яких визначено спосіб з'єднання країв один з одним. Лабіринт на поверхні куба - це всього лише шість квадратних частин лабіринту. Коли створювана частина доходить до краю, то вона перетікає в наступну частину і в правий край.
Шаблон	лабіринти, засновані на шаблонах, створюються, починаючи з базового зображення-шаблону, а потім запускається видалення ізольованих областей, що забезпечує наявність рішення лабіринту, за яким слід видалення петель, що підвищує складність лабіринту. В результаті створюється ідеальний лабіринт, дуже схожий на вихідне зображення.

Таблиця 2.3.2 – класифікація алгоритмів побудови лабіринтів

2.3.3 Алгоритми створення ідеальних лабіринтів

Для побудови ідеальних лабіринтів існує безмежна кількість методів зі своєю класифікацією. У таблиці 2.3.3 наведено список найбільш поширених алгоритмів.

Recursive backtracker	він у чомусь схожий на метод вирішення лабіринтів recursive backtracker, і вимагає стека, обсяг якого може доходити до розмірів лабіринту. При вирізанні він поводиться максимально жадібно, і завжди вирізає прохід в нествореній частині, якщо вона існує поруч з поточної осередком.
Алгоритм Краскала	це алгоритм, який створює мінімальний сполучна дерево. Це цікаво, тому що він не «вирощує» лабіринт подібно дереву, а скоріше вирізає сегменти проходів по всьому лабіринту випадок-ним чином, і тим не менш в результаті створює в кінці ідеальний лабіринт. Для його роботи потрібно обсяг пам'яті, пропорційний раз-міру лабіринту, а також можливість перерахування кожного ребра або стіни між осередками лабіринту у випадковому порядку (зазвичай для цього створюється список усіх ребер і перемішується випадковим чином).
Алгоритм Прима (істинний)	цей алгоритм створює мінімальне дерево, обробляючи унікально випадкові ваги ребер. Обсяг требуєм пам'яті пропорційний розміру лабіринту.

Алгоритм Прима (спрощена)	цей алгоритм Прима створює мінімально сполучна дерево. Він спрощений таким чином, що всі ваги ребер однакові. Для нього потрібно обсяг пам'яті, пропорційний розміру лабіринту.
Алгоритм Прима (модифікований)	цей алгоритм Прима створює мінімальне сполучна дерево, змінений так, що всі ваги ребер одна підступи. Однак він реалізований таким чином, що замість ребер дивиться на осередки. Об'єм пам'яті пропорційний розміру лабіринту.
Алгоритм Олдоса-Бродера	цікаво в цьому алгоритмі то, що він однорідний, тобто він з однаковою ймовірністю створює всі можливі лабирінту заданого розміру. Крім того, йому не потрібно додаткової пам'яті або стека.
Алгоритм Уїлсона	це вдосконалена версія алгоритму Олдоса-Бродера, він створює лабіринти точно з такою ж текстурою (алгоритми однорідні, тобто всі можливі лабіринти генеруються з рівною ймовірність), проте алгоритм Уїлсона виконується набагато швидше. Він займає пам'ять аж до розмірів лабіринту.
Алгоритм Hunt and kill	цей алгоритм зручний, тому що не вимагає додаткової пам'яті або стека, а тому підходить для створення величезних лабіринтів або лабіринтах на слабких комп'ютерах завдяки НЕ-можливості вичерпання пам'яті. Так як в ньому немає правил, яким нуж-но слідувати постійно, його також найпростіше модифікувати з його допомогою лабіринти з різною текстурою.

дерева (Growing tree algorithm)	це узагальнений алгоритм, здатний створювати лабіринти з різною текстурою. Необхідна пам'ять може досягнень-гать розміру лабіринту.
Алгоритм вирощування лісу (Growing forest algorithm)	це більш узагальнений алгоритм, що поєднує в собі типи, засновані на деревах і множини. Він є розширенням алгоритму вирощування дерева, по суті включає в себе кілька примірників, що розширюються одночасно. Починаємо з усіх осередків, випадковим чином впорядкований-них в список «нових»; крім того, у кожного осередку є власне моно дружність, як на початку алгоритму Краскала.
Алгоритм Еллера	це особливий алгоритм, тому що він не тільки швидше всіх інших, але і не має очевидної смещённості або недоліки; крім того, при його створенні пам'ять використовується найбільш ефективний. Для нього навіть не потрібно, щоб в пам'яті знаходився весь лаби-Рінту, він використовує обсяг, пропорційний розміру рядка.
Рекурсивне розподіл (Recursive division)	цей алгоритм чимось схожий на recursive backtracking, тому що в них обох застосовуються стеки, тільки він працює не з проходами, а зі стінами.
Лабіринти на основі довічних дерев	по суті, це найпростіші і швидкі з можливих алгоритмів, однак створювані лабіринти мають текстуру з дуже високою смещённостью.
Лабіринти Sidewinder	цей простий алгоритм дуже схожий на алгоритм двійкового дерева, але трохи більш складний.

Таблиця 2.3.2 – класифікація алгоритмів побудови ідеальних лабіринтів.

2.4 Алгоритм Еллера

Алгоритм Еллера — дозволяє процедурно генерувати одно маршрутний ідеальний лабіринт який не містить у собі циклів. Сереед інших алгоритмів, важається найоптимальнішим через швидкий процес генерації, та невеликий об’єм пам’яті для зберігання даних про лабіринт (пропорційну довжені рядка лабіринту). Через специфіку запам’ятання лише останнього рядка, дозволяє генерувати нескінченно великий у довжину лабіринт.

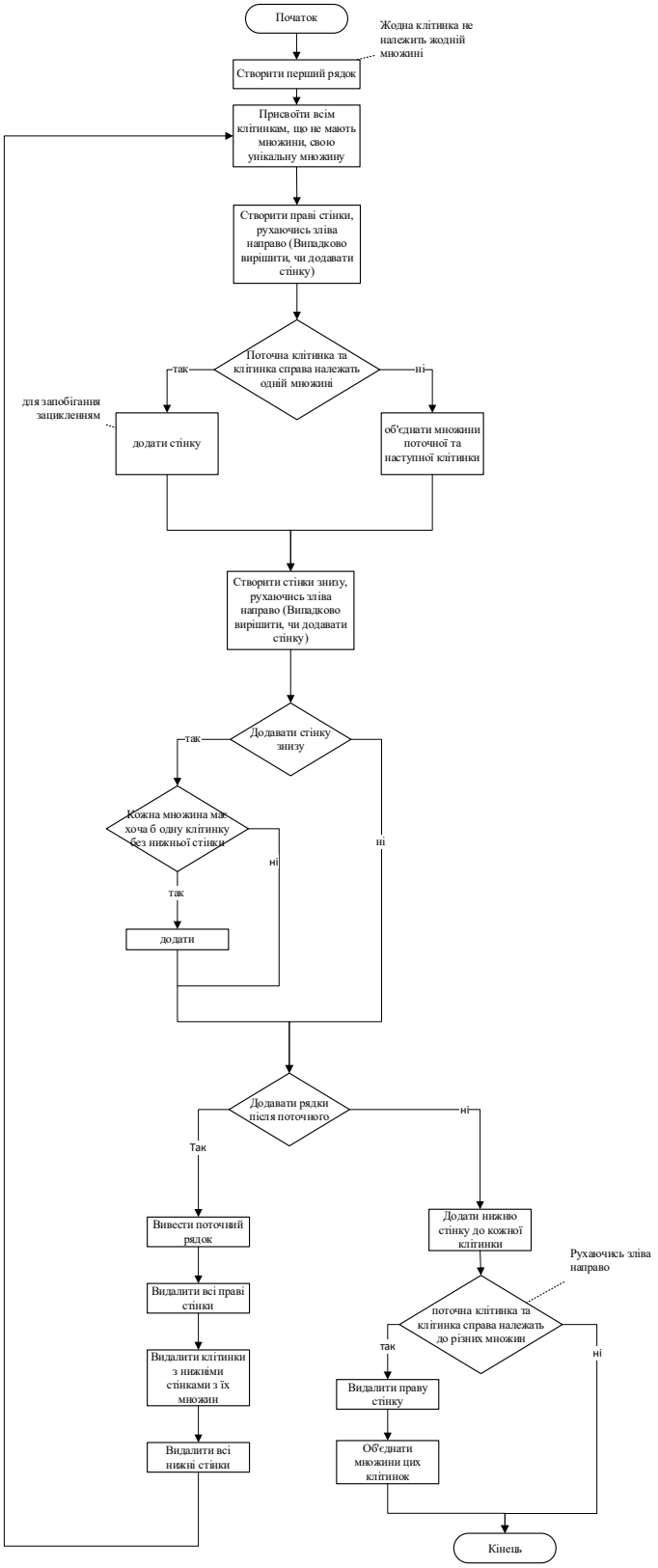


Рисунок 2.4 – Алгоритм Еллера

3. АНАЛІЗ ВАРІАНТІВ СТВОРЕННЯ ЛАБІРИНТІВ ЗА ДОПОМОГОЮ ФРАКТАЛІВ

3.1 Використання фракталів при побудовіі лабіринтів

Часто фрактали використовують для побудови лабіринтів у грі. В загальному лабіринти можна поділити на двовимірні, трьох вимірні, nвимірні та переплетіння. На рисунку 1 наведено розподіл фрактальних лабіринтів, що використовуються в відеоіграх. Фраткальні лабіринти засновуються на випадковому числу, яке відповідає за фрактальну гомотетію лабіринту.

Застосування 3D фракталів в віртуальній реальності продемонстровано у грі Marble Marcher це ігровий простір створено єдиним алгоритмом. В наведеній грі всі об’єкти підпорядковуються математичним залежностям.

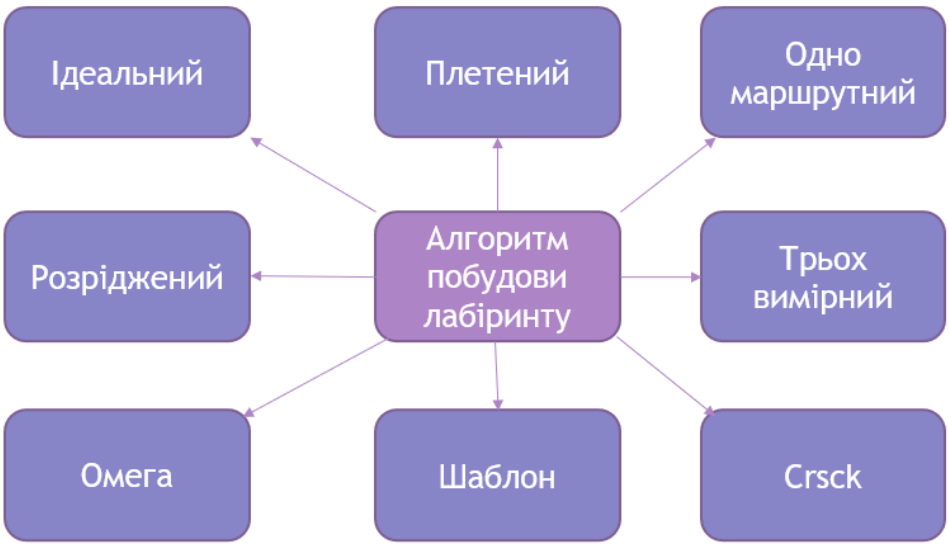


Рис.3.1.1 Класифікація алгоритмів побудови лабіринту

Віртуальні фрактальний лабіринт (ВФЛ) - лабіринт, який не зберігається в пам'яті одночасно. Він може зберігати тільки частину проходів розміром 100x100 поруч з вашим місцем розташування в симуляції безкінечно великого лабіринту. Для побудови величезний віртуальних лабіринтів використовують розширення вкладених фрактальних лабіринтів.

Самі ВФЛ можна порівняти з фракталом Мандельброта, адже в своїх зображеннях він існує віртуально, та умовою його проявлення є перегляд певної координати при досить високому збільшені.

Є безліч варіантів реалізації фрактальних лабіринтів, але великої популярності набули метод рекурсивного поділу та створення лабіринту на базі вже існуючого фракталу.

Алгоритм “рекурсивного поділу” – за пріоритетністю відноситься до «додавання стін». Його фрактальну структуру гарно видно при високому рівні деталізації. По суті своєї процес побудови лабіринту можна продовжувати нескінченно довго, поглибжуючи рівень деталізації.

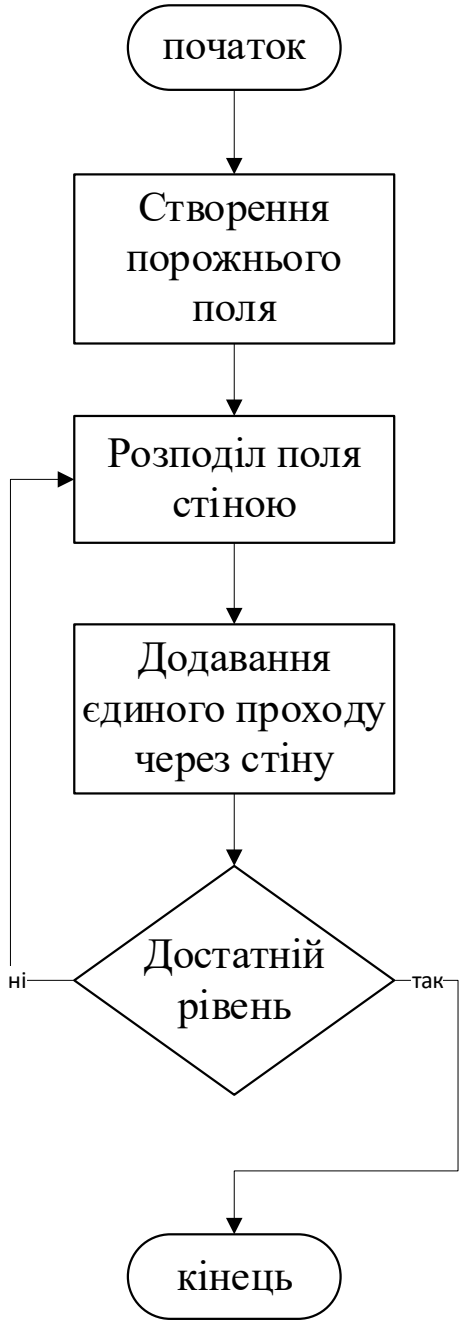


Рисунок 3.1.2 Рекурсивний метод

Як можна помітити, фактично на кожній поточній ітерації ми отримуємо діючий лабіринт, а з кожною наступною ми збільшуємо його деталізацію.

Розглянемо цей метод більш детально на основі нескладного прикладу:

- Невелику сітку 7 × 7, розділимо її по вертикалі при x = 5;
- Випадковим чином робимо отвір у стіні;

- По завершенні ітерації, переходимо до правої (меньщої) частини поля відносно стіни;
- Ділимо поле горизонтально, за для запобігання створення вузьких проходів, при довільному y;
- Випадковим чином робимо отвір;
- Переходимо, до нижньої (меньщої) частини, та повторюємо дії;
- При умові того що розмір секції поточного біполя рівний розміру отвору, подальший дії неможливі, тому повертаємося на крок назад та продовжуємо рекурсивний поділ.

Важливо зауважити, що ширина проходів має обов’язково відповідати розміру отворів у стінах, що по суті своєї, є показником глибини рекурсії (при сталому розмірі початкового поля), також різниця між сторонами поточного біполя є умовою вибору способу поділу поля, наприклад, якщо ширина поля більша за висоту, то ділимо вертикально. Для створення цікавих варіантів згенерованих лабіринтів, надають перевагу вертикальному поділу та випадковому числі співвідношення сторін при ньому.

У результаті роботи алгоритму рекурсивного ділення ми отримуємо ідеальний лабіринт.

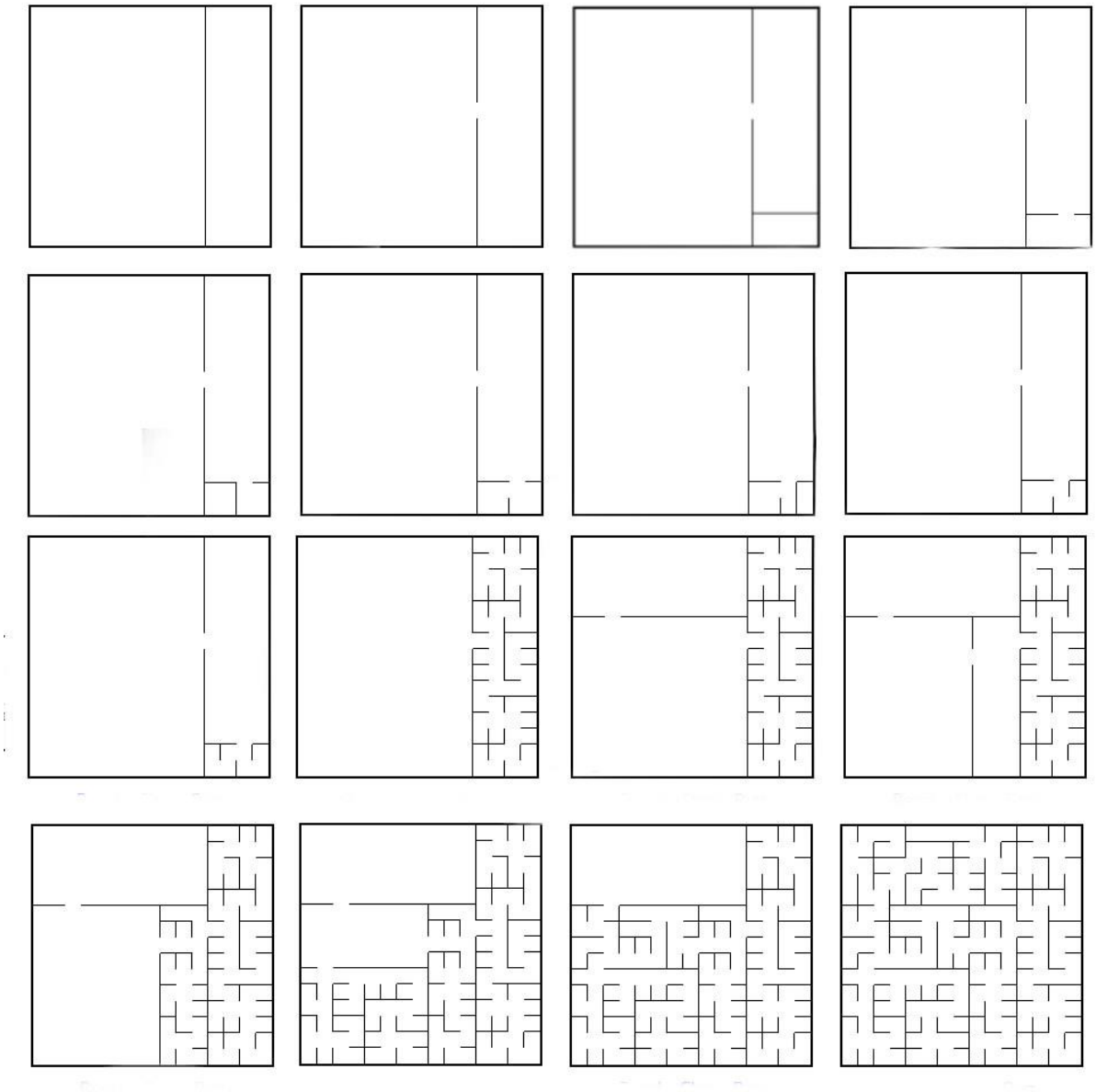


Рисунок 3.1.3 – Приклад роботи алгоритму рекурсивного ділення

Другий метод побудови фрактального лабіринту розглянемо на основі фракталу килим Серпінського.

Килим Серпінського (КС) — плоский фрактал, вперше описаний Вацлавом Серпінським в 1916 році. Одна із варіацій множини Кантора у двох вимірах. Є універсальною кривою, де будь-який можливий одномірний граф, спроектований на двовимірну площину, також гомеоморфний до підмножини серветки Серпінського.

Побудова КС починається з рівностороннього прямокутника. Його розділяють на 9 рівних часнин, утворюючи тим самим сітку три на три, при цьому центральний підквадрат видаляється. З кожним наступним кроком ітерації, необхідно провести такі самі дії з новоутвореними підквадратами.

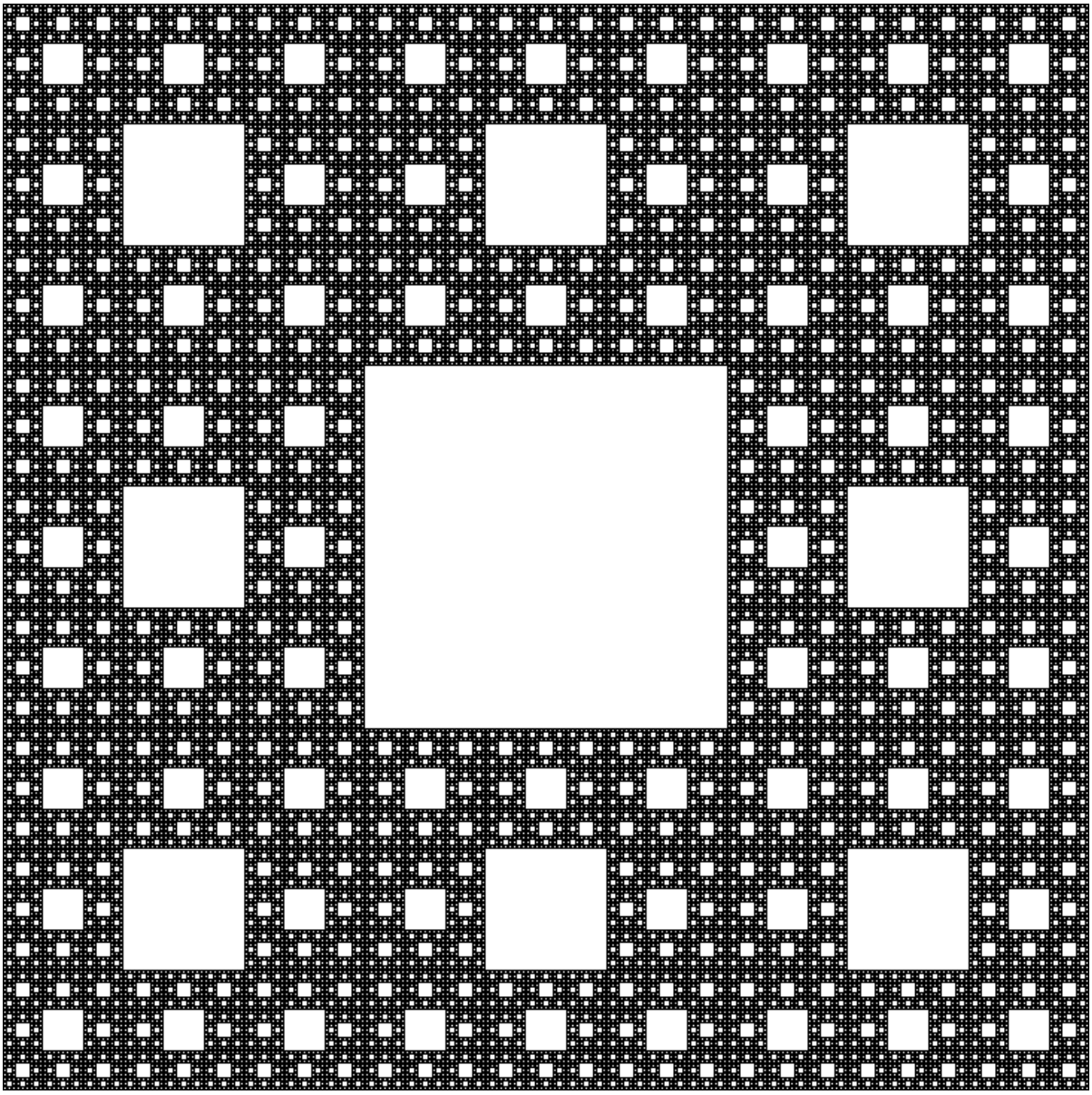


Рисунок 3.1.4 – Килим Серпінського

Метод хаоса:

- Задаються 8 точок, вершини та середини сторін початкового квадрату Q_0

- Ймовірний простір (0;1) розбивається на 8 осередків.
- Задається деяка початкова точка $P_0 \in Q_0$.
- Початок цикла
  - Генеруємо випадкове число $n \in (0;1)$.
  - Активним аттрактором стає та вершина, на розподіл усіх підпростір якої випало згенероване число.
  - будується точка P_i з новими координатами: $x_i = \frac{x_{i-1}+2x_A}{3}$; $y_i = \frac{y_{i-1}+2y_A}{3}$, x_{i-1}, y_{i-1} — координати P_i, x_A, y_A — координати активної точки-аттрактора.
  - Повертаємося до початку цикла.

3.2 Недоліки та переваги використання фрактталів в відеоіграї

Фрактальна графіка позбавлена більшості недоліків растрової та векторної графіки. Наведемо таблицю недоліків та переваг використання фракталів у відеоіграх. Вони наведені у таблиці 3.2

Переваги	Недоліки
Не великий об’єм даних (зберігаються лише алгоритми)	Непередбачувана поведінка об’єкту при зміні параметра системи
Деталізація об’єкту	Складні математичні формули для відтворення віртуальних та реальних об’єктів

Можливість відтворення з частини об’єкту повністю весь об’єкт	Складні зображення перевантажуються ЦП та ОЗУ
Простота в модифікації (зміна лише одного параметру системи може повністю змінити об’єкт)	Не підтримка існуючих програмних розробок для створення фрактальної графіки різними операційними системи
Можливість відтворення реальних об’єктів	Обмеженість початкових фігур об’єкту
Об’ємність зображення	Недостатня кількість спеціалістів в області фрактальної графіки

Таблиця 3.2 – порівняння переваг та недоліків

4. ЗАСОБИ РОЗРОБКИ

4.1 Обґрунтування вибору технологій та їх короткий опис

З розвитком ігрової індустрії з’явилося багато інструментів для реалізації тих чи інших проектів, до таких можна віднести: бібліотеку SFML, ігровий рушій MonoGame, Love 2D, Godot, Unreal Engine та Unity. З них найбільш доступним є останій.

4.2 Опис кросплатформленого ігрового двигуну Unity

Unity — багатоплатформовий інструмент для розробки відеоігор, а також рушій, на якому вони працюють. Програми що було створені на базі Unity можуть вільно запускатися як на ПК, так й на мобільних пристроях чи ігрових консолях та на пристроях віртуальної чи доповненої реальності. Також Вони підтримують 2Д, 3Д графіку.

Ігрова логіка пишеться на основі мови C#. Unity підтримує редагування скриптів у середовищі Visual Studio.

Рендеринг зображення	відбувається через віртуальну камеру огляду. В робочій області редактора ігрова сцена може розміщуватися як завгодно, а

	при рендерингу — так, як її видно з камери. В сцені може бути декілька камер, які рухаються за персонажем чи за вказаною траєкторією. Вигляд з камери подається в двовимірно чи тривимірно (в перспективі або ортографічно). Фон сцени, видимий через камеру, типово зображає небо, утворене скайбоксом, але може презентувати й інше довкілля.
Графічний рушій	використовує DirectX (Windows), OpenGL (Mac, Windows, Linux), OpenGL ES (Android, iOS), та спеціальне власне API для Wii. Також підтримуються bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), динамічні тіні з використанням shadow maps, render-to-texture та повноекранні ефекти post-processing.
Написання шейдерів	використовується ShaderLab, що підтримує шейдерні програми написані на GLSL або Cg. Шейдер може включати декілька варіантів реалізації, що дозволяє Unity визначати найкращий варіант для конкретної відеокарти. Unity також має вбудовану підтримку фізичного рушія Nvidia PhysX (колишнього Ageia), підтримку симуляції одягу в системі реального часу на довільній та прив'язаній полігональній сітці (починаючи з Unity 3.0),

	підтримку системи ray casts та шарів зіткнення.
Скриптова система ігрового рушія	зроблена на Mono — вільний відкритий проєкт з реалізації .NET Framework. Програмісти можуть використовувати UnityScript (власна скриптова мова, подібна до JavaScript та ECMAScript), C# або Boo (мова програмування, подібна до Python). Починаючи з версії 3.0, до Unity входить перероблена версія MonoDevelop для зневадження скриптів.
Система контролю версій	для ігрових об'єктів та скриптів під назвою Unity Asset Server. Система використовує PostgreSQL, роботу зі звуком, побудовану на основі бібліотеки FMOD (з можливістю програвати Ogg Vorbis аудіофайли), відеопрогравач із кодеком Theora, рушій для побудови ландшафтів рослинності, вбудовану систему карт освітлення (Beast), мережу для мультиплеєру (RakNet) та вбудовані навігаційні меші для пошуку шляху.
Сервер наборів ресурсів	платне доповнення, що додає інструментарій для спільної розробки на базі Unity багатьома користувачами одночасно та контроль версій у функціоналі Unity.
Підтримка файли	3ds Max, Maya, Softimage, Blender, modo, ZBrush, Cinema 4D, Cheetah3D, Adobe Photoshop, Adobe Fireworks та Allegorithmic Substance. В ігровий проєкт Unity можна

	імпортувати об'єкти цих програм та виконувати налаштовування за допомогою графічного інтерфейсу.

Таблиця 4.2 – основні характеристики роботи Unity

До переваг Unity можна віднести:

- Багатогігабайтні проєкти з тисячами мегабайтних файлів піддаються легкому керуванню.
- Налаштування імпорту та інші метадані також зберігаються разом з історією їх версій.
- Переглядати зміни ресурсів\версій можна одразу всередині редактора Unity. Якщо файли змінюються, їх статус негайно оновлюється.
- Перейменування і переміщення ресурсів не створює будь-яких перешкод для безперервного робочого процесу. Сервер ресурсів Unity управляється базою даних PostgreSQL.
- Сервер ресурсів доступний як для Mac OS X Installer, так і для Linux RPMs . Підтримка декількох платформ забезпечує гнучкість у впровадженні Сервера ресурсів Unity у наявну IT-інфраструктуру.

4.3 Опис мови програмування C#

C# (вимовляється Сі-шарп) — об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research.

Синтаксис C# близький до C++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML.

5. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

5.1 Архітектура програми

Основна програма складається з трьох модулів: модулю управління ігровим персонажем, модуль побудови оточення, модуль управління графічної складової.

5.2 Опис програмної реалізації

```
// where will the wall be drawn from?
int wx = x + (horizontal ? 0 : Random.Range(0, width - 2));
int wy = y + (horizontal ? Random.Range(0, height - 2) : 0);

// where will the passage through the wall exist?
int px = wx + (horizontal ? Random.Range(0, width) : 0);
int py = wy + (horizontal ? 0 : Random.Range(0, height));

// what direction will the wall be drawn?
int dx = horizontal ? 1 : 0;
int dy = horizontal ? 0 : 1;

// how long will the wall be?
int length = horizontal ? width : height;
```

Початок роботи з рекурсивним методом.

Задання випадкових основніх точок для побудови стіни та проходів: wx, wy, px, py.

Додавання кілька допоміжних значень, які допомагають намалювати стіну: dx, dy, length.

```
// what direction is perpendicular to the wall?
int dir = horizontal ? S : E;

for (int i = 0; i < length; i++)
{
    if (wx != px || wy != py)
    {
        board[wy, wx] |= dir;
    }

    wx += dx;
    wy += dy;
}

int nx = x;
int ny = y;

int w = horizontal ? width : wx - x + 1;
int h = horizontal ? wy - y + 1 : height;

Divide(board, nx, ny, w, h, choose_orientation(w, h));

nx = horizontal ? x : wx + 1;
ny = horizontal ? wy + 1 : y;

w = horizontal ? width : x + width - wx - 1;
h = horizontal ? y + height - wy - 1 : height;

Divide(board, nx, ny, w, h, choose_orientation(w, h));
```

Одразу після задання основних налаштовань,в циклі for малюємо стіну. Визначаємо межі частин біполя, та повторюємо діїї.

```
public void DisposeOldMaze()
{
    GameObject[] objects = GameObject.FindGameObjectsWithTag("Generated");
    foreach (GameObject go in objects)
    {
        Destroy(go);
    }
}
```

DisposeOldMaze() видаляє існуючий лабіринт

```
private void GenerateFractal(int startX, int startY, int blockSize)
{
    if (blockSize < 1)
        return;

    for (int x = startX + blockSize; x < startX + 2 * blockSize; x++)
    {
        for (int y = startY + blockSize; y < startY + 2 * blockSize; y++)
        {
            board[x, y] = true;
        }
    }

    for (int x = 0; x < 3; x++)
    {
        for (int y = 0; y < 3; y++)
        {
            int newBlockSize = blockSize / 3;
            GenerateFractal(startX + blockSize * x, startY + blockSize * y, newBlockSize);
        }
    }
}
```

Метод створення лабіринту на базі КС

У першому циклі йде розбиття підполе на 9 сигментів та побудова основного центрального блоку (куб на 5-му сигменті).

Другий цикл зменьшує розмір блоку у три рази, та рекурсивно звертається до функції, переходячи на нову ітерацію побудови фрактала.

```
using System;
using UnityEngine;
using UnityEngine.UI;

[RequireComponent(typeof(MazeConstructor))]

public class GameController : MonoBehaviour
{
    //1
    [SerializeField] private FpsMovement player;
    [SerializeField] private Text timeLabel;
    [SerializeField] private Text scoreLabel;

    private MazeConstructor generator;

    //2
    private DateTime startTime;
    private int timeLimit;
    private int reduceLimitBy;

    private int score;
    private bool goalReached;
```

1. Серіалізовані поля для об'єктів в сцені.
2. Приватних змінних для відстеження таймера і очок гри, а також того, знайдена чи мета в лабіринті.

```
    //3
    void Start()
    {
        generator = GetComponent<MazeConstructor>();
        StartNewGame();
    }

    //4
    private void StartNewGame()
    {
        timeLimit = 80;
        reduceLimitBy = 5;
        startTime = DateTime.Now;

        score = 0;
        scoreLabel.text = score.ToString();

        StartNewMaze();
    }
```

3. MazeConstructor визиває методи побудови лабіринту

4. StartNewGame () використовується для запуску всієї гри спочатку, а не для перемикання рівнів всередині гри. Таймером присвоюються початкові значення, окуляри скидаються, після чого створюється лабіринт.

```
//5
private void StartNewMaze()
{
    generator.GenerateNewMaze(13, 15, OnStartTrigger, OnGoalTrigger);

    float x = generator.startCol * generator.hallWidth;
    float y = 1;
    float z = generator.startRow * generator.hallWidth;
    player.transform.position = new Vector3(x, y, z);

    goalReached = false;
    player.enabled = true;

    // restart timer
    timeLimit -= reduceLimitBy;
    startTime = DateTime.Now;
}
```

5. StartNewMaze () переходить до нового рівня, що не перезапуску заново всю гру. Крім створення нового лабіринту, цей метод має гравця в початковій точці, скидає мета і знижує ліміт часу.

```
//6
void Update()
{
    if (!player.enabled)
    {
        return;
    }

    int timeUsed = (int)(DateTime.Now - startTime).TotalSeconds;
    int timeLeft = timeLimit - timeUsed;

    if (timeLeft > 0)
    {
        timeLabel.text = timeLeft.ToString();
    }
    else
    {
        timeLabel.text = "TIME UP";
        player.enabled = false;

        Invoke("StartNewGame", 4);
    }
}
```

6. Update () перевіряє, чи активний гравець, а потім оновлює час, що залишився на проходження рівня. Після завершення часу гравець деактивується і починається нова гра.

```
//7
private void OnGoalTrigger(GameObject trigger, GameObject other)
{
    Debug.Log("Goal!");
    goalReached = true;

    score += 1;
    scoreLabel.text = score.ToString();

    Destroy(trigger);
}

private void OnStartTrigger(GameObject trigger, GameObject other)
{
    if (goalReached)
    {
        Debug.Log("Finish!");
        player.enabled = false;

        Invoke("StartNewMaze", 4);
    }
}
```

7. OnGoalTrigger () і OnStartTrigger () - це функції обробки подій, що передаються TriggerEventRouter в MazeConstructor. OnGoalTrigger () записує, що мета була знайдена, а потім збільшує кількість очок. OnStartTrigger () перевіряє, знайдена чи мета, і якщо це так, то деактивує гравця і запускає новий лабіринт.

6. РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

6.1 Системні вимоги для додаткове програмне забезпечення

Програмний продукт сам по собі не потребує додаткових розширень, та може вільно запускатися на ПК. У таблиці 6.1 низче наведені основні вимоги до ПК

Операцыйна система	Windows 10
Процесор	Intel(R) Core(TM) i3-6006U CPU @ 2.00GHz 1.99 GHz
Оперативна пам’ять	8,00 ГБ
Тип системи	64-розрядна операцыйна система, процесор x64
Вільне місце на диску	100 мб

Таблиця 6.1 – вимоги до ПК

6.2 Результати виконання програми

Ціль гри полягає у пошуку жовтих коробок всередині замкнутого фрактального лабіринту за відведений час.

При старті гри, гравець опиняється у випадковому місці згенерованої карти мапи (рисунок 6.2.1), у лівому верхньому куті відображається кількість підібраних коробок (рисунок 6.2.2-6.2.3), у правому куті – таймер. На пошук

дається 80 секунд. По завершенню гри, при натискані на кнопку «enter», вона почнеться знову з самого початку, проте вже в новому лабіринті

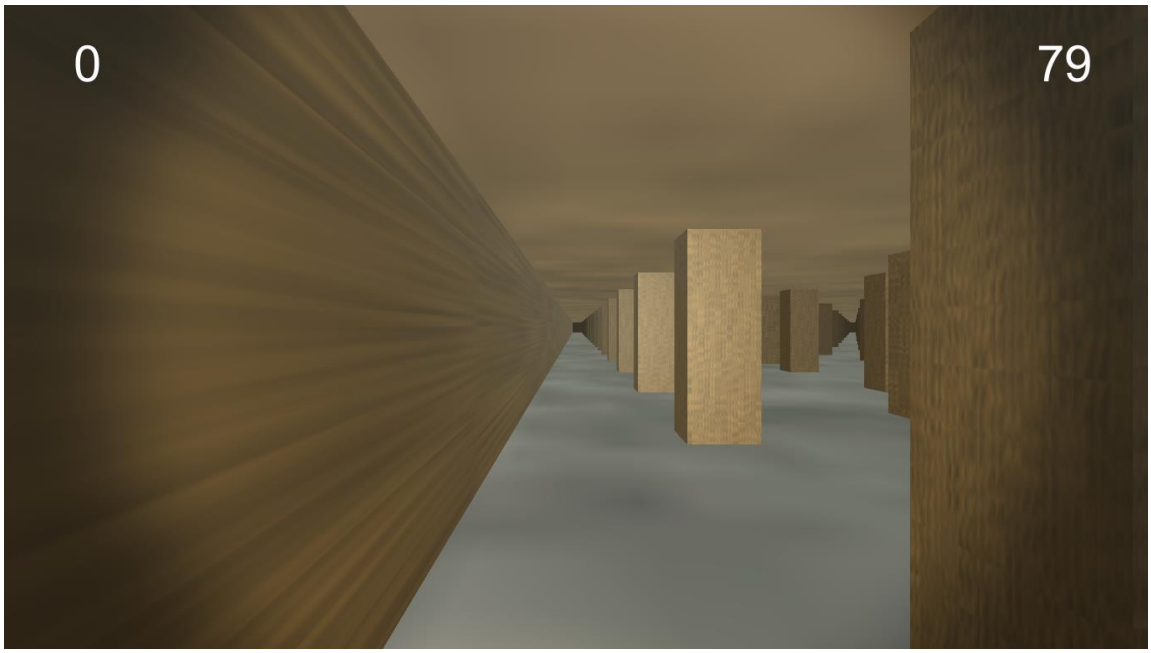
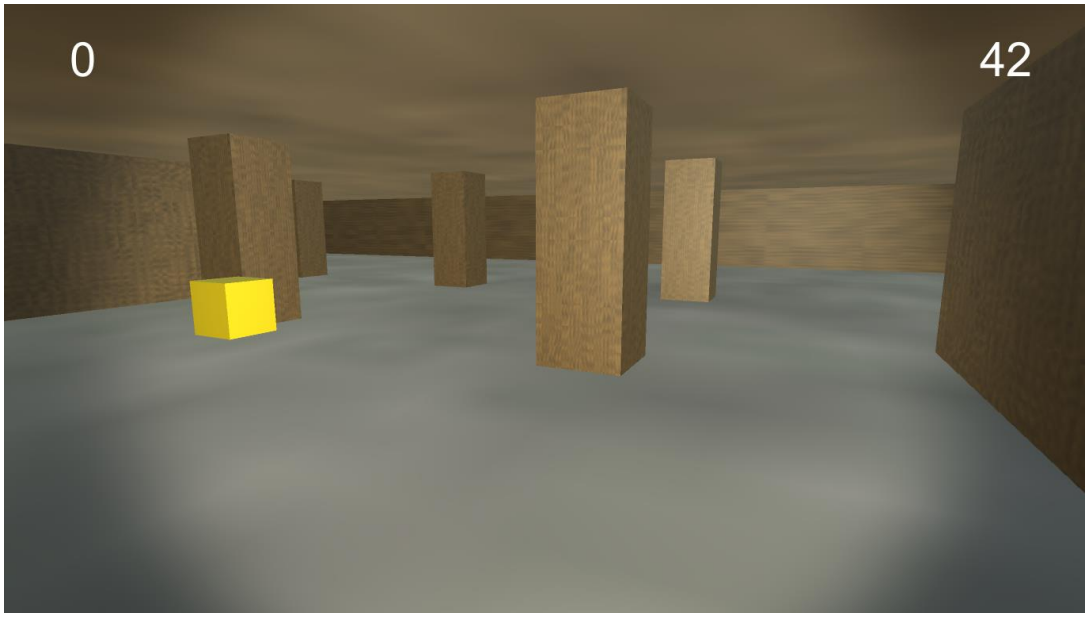


Рисунок 6.2.1 – Початок гри.



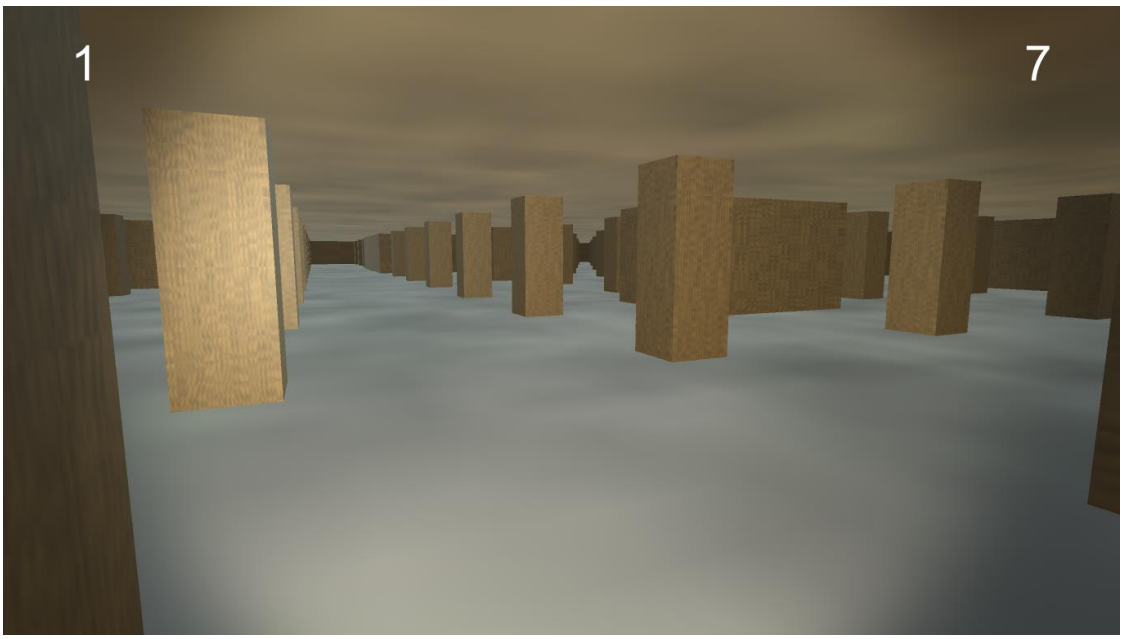


Рисунок 6.2.2-6.2.3 – Зміна лічильника після знаходження коробки

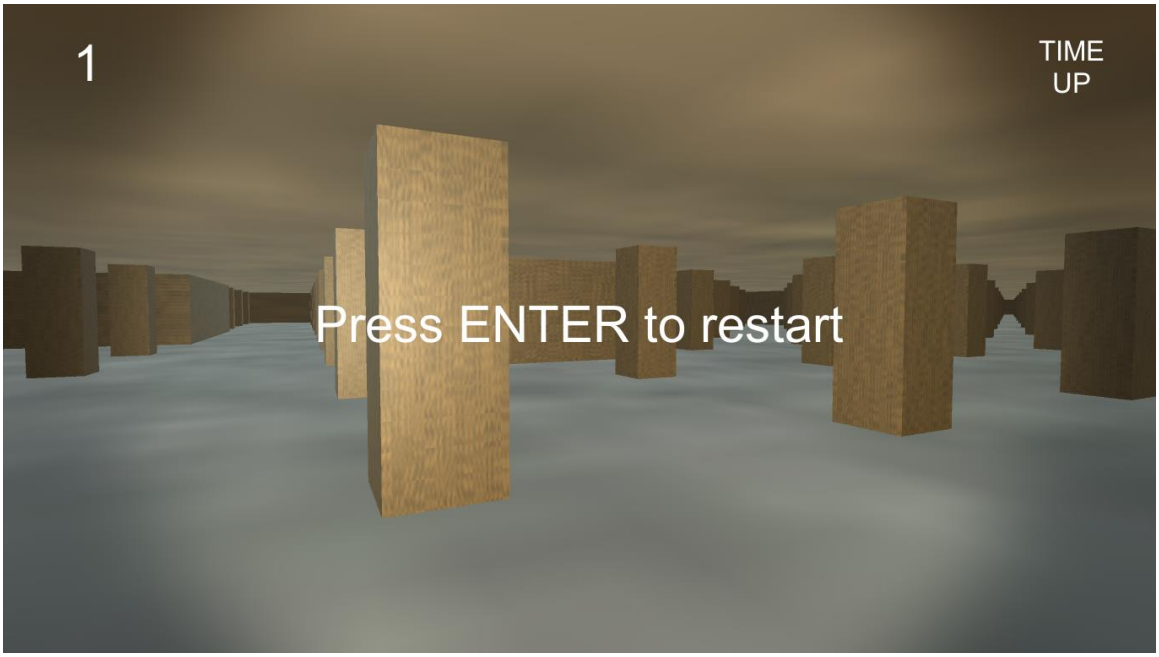


Рисунок 6.2.4 – Завершення гри

Оскільки при кодному новому запуску гри, вона випадково генерує одиу з двох можливих варіацій фрактального лабіринту, з точки боку розробника ми можемо побачити наступні варіанти генецації.

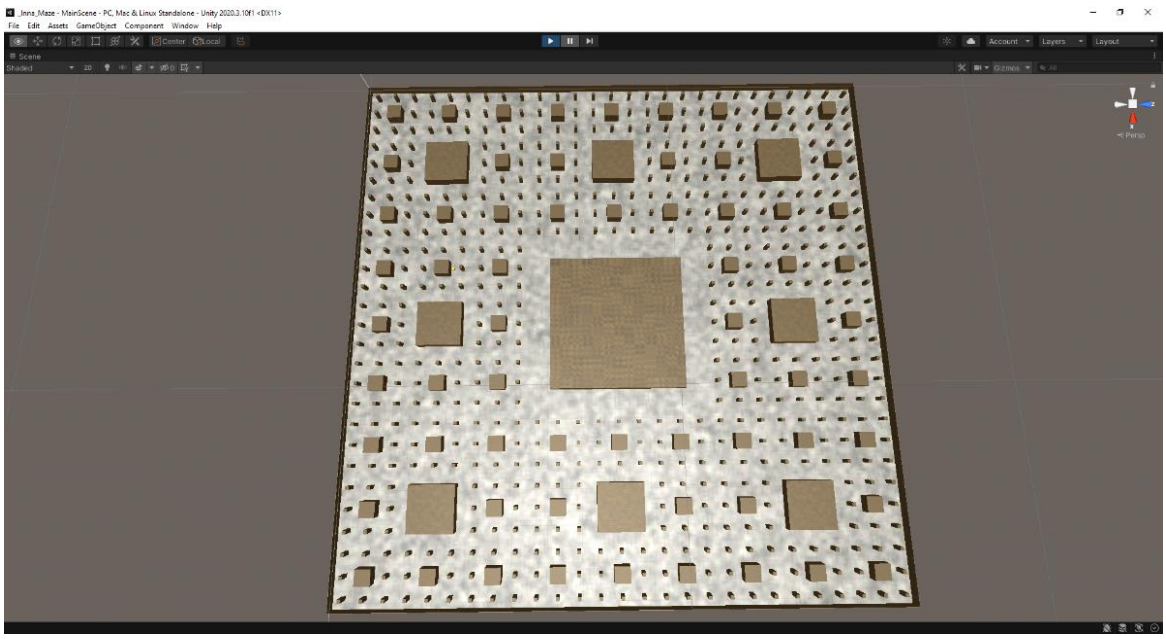


Рисунок 6.2.5 – варіант реалізації КС

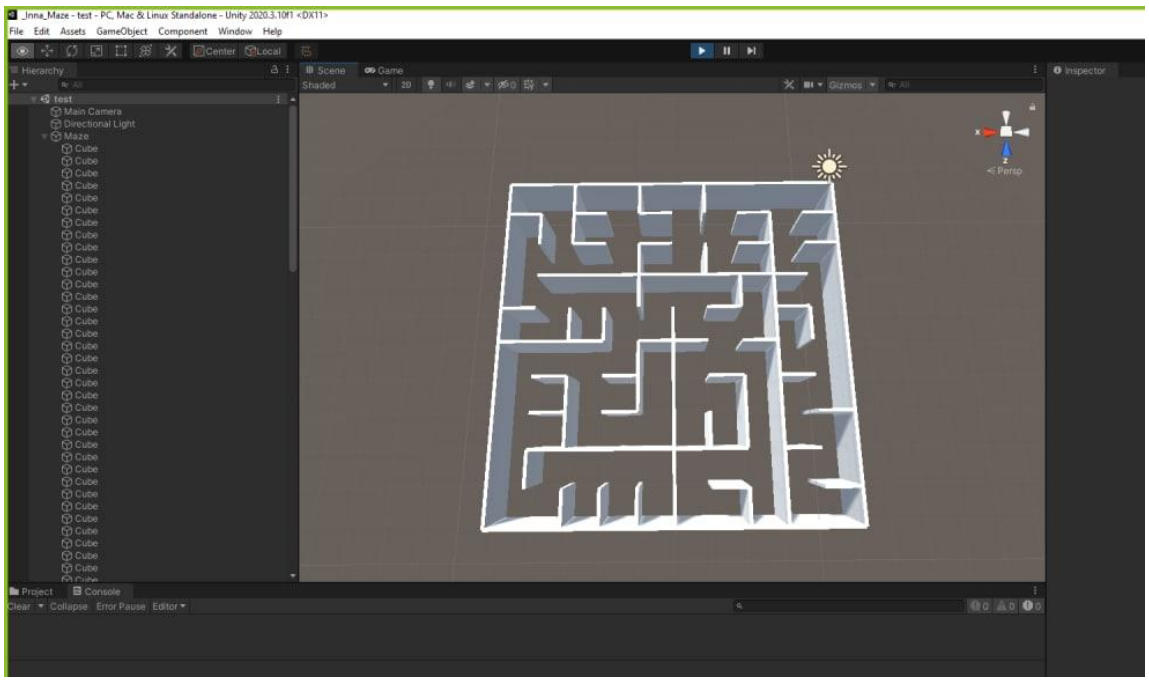
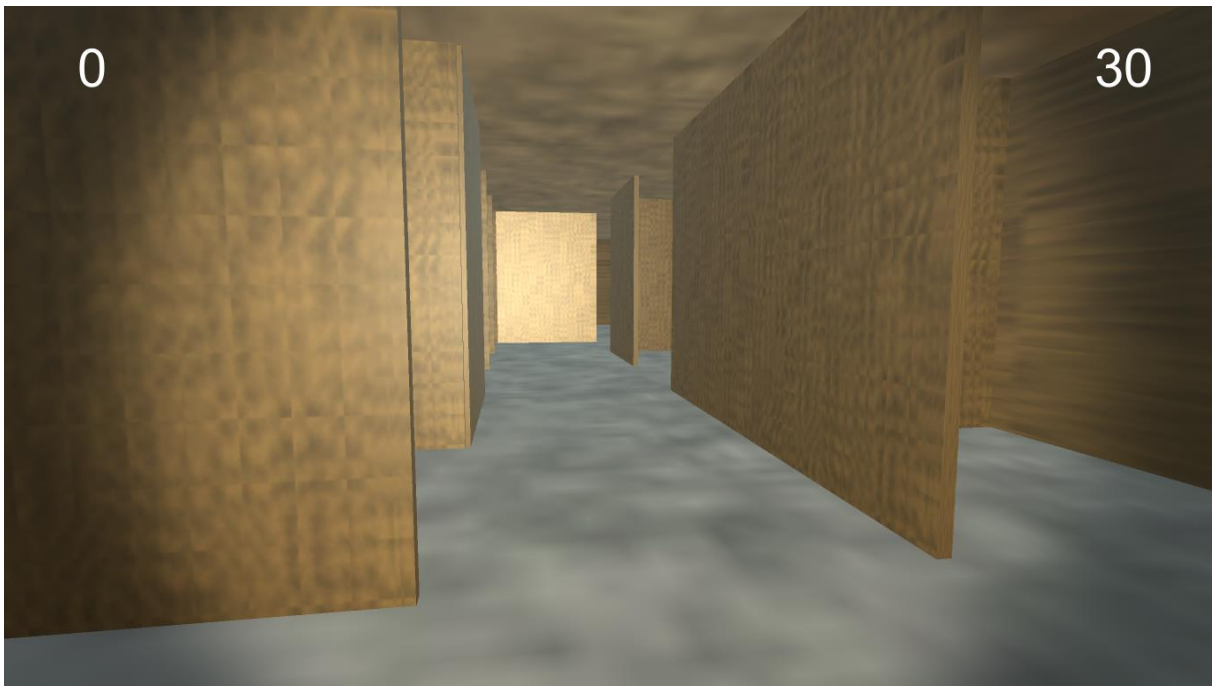
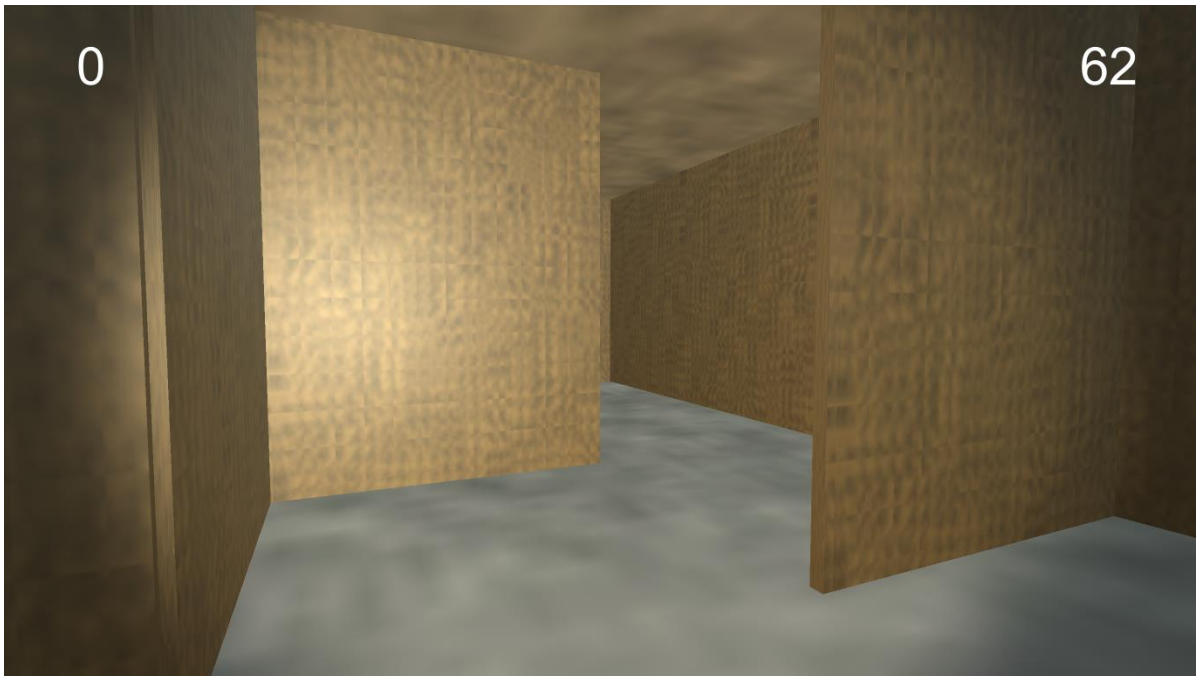


Рисунок 6.2.6 – Варіант реалізації методу рекурсивного поділу

Вигляд лабіринту на основі КС, очима користувача було продемонстровано на рисунках 6.2.1-6.2.4.

Лабірин згенерований методом рекурсивного ділення продемонстровано на рисунках низче.


0
30


0
62

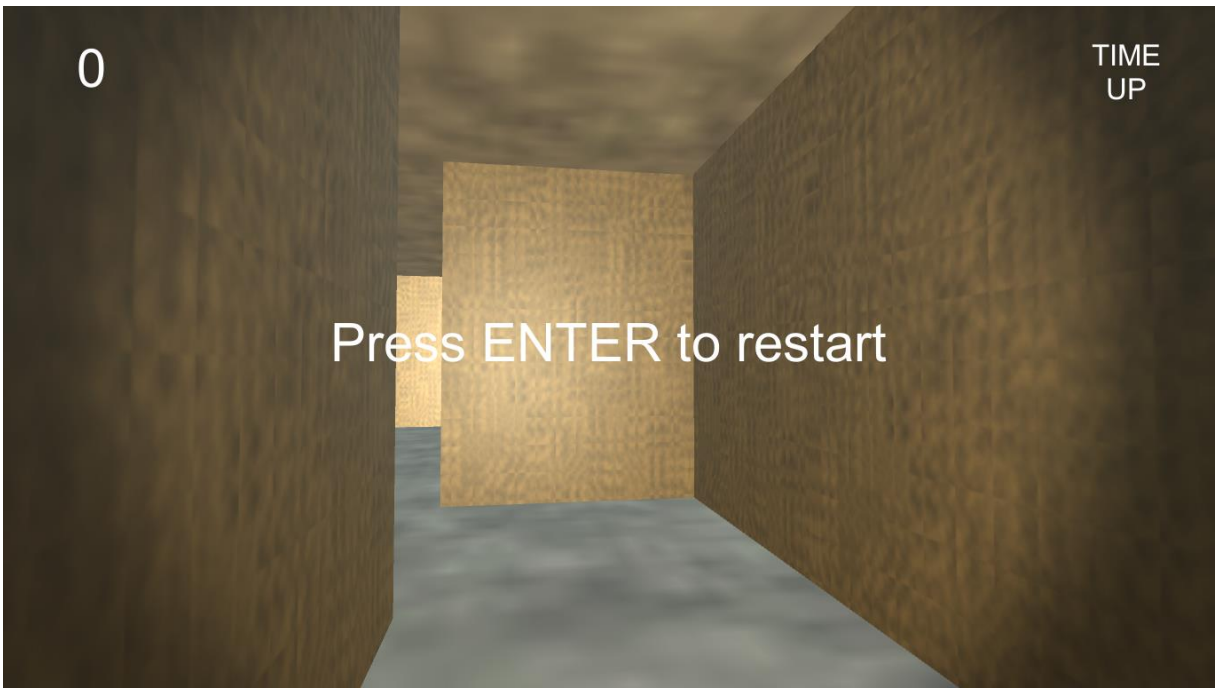


Рисунок 6.2.7-6.2.9 – лабіринт побудований рекурсивним діленням з точки зору користувача

ВИСНОВКИ

Проведений аналіз існуючих відеоігр вказав на ряд недоліків та переваг при використання фракталів для моделювання ігрового процесу. Зважаючи на огляд ігрових застосунків можливо визначити сфери в яких застосування фракталів буде оптимальним. До таких сфер використання наделать ігрова графіка (як реальна, так і віртуальна), взаємозв’язок між персонажами, ігри в яких задіяно віртуальні лабіринти та такі стратегія яких полягає у «частинка подібна цілому». Використання фракталів розширює клас графічних явних та неявних об’єктів та функціональні можливості гри.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

7. Джон Скит. C# для профессионалов: тонкости программирования, 3-е издание, новый перевод = C# in Depth, 3rd ed.. — М.: «Вильямс», 2014. — 608 с. — ISBN 978-5-8459-1909-0.
8. Кристиан Нейгел и др. C# 5.0 и платформа .NET 4.5 для профессионалов = Professional C# 5.0 and .NET 4.5. — М.: «Диалектика», 2013. — 1440 с. — ISBN 978-5-8459-1850-5.
9. А. Хейлсберг, М. Торгерсен, С. Вилтамут, П. Голд. Язык программирования C#. Классика Computers Science. 4-е издание = C# Programming Language (Covering C# 4.0), 4th Ed. — СПб.: «Питер», 2012. — 784 с. — ISBN 978-5-459-00283-6. Архивная копия от 10 октября 2011 на Wayback Machine
10. Э. Стиллмен, Дж. Грин. Изучаем C#. 2-е издание = Head First C#, 2ed. — СПб.: «Питер», 2012. — 704 с. — ISBN 978-5-4461-0105-4. (недоступная ссылка)
11. Эндрю Троелсен. Язык программирования C# 5.0 и платформа .NET 4.5, 6-е издание = Pro C# 5.0 and the .NET 4.5 Framework, 6th edition. — М.: «Вильямс», 2013. — 1312 с. — ISBN 978-5-8459-1814-7.
12. Джозеф Албахари, Бен Албахари. C# 6.0. Справочник. Полное описание языка = C# 6.0 in a Nutshell: The Definitive Reference. — М.: «Вильямс», 2018. — 1040 с. — ISBN 978-5-8459-2087-4. — ISBN 978-1-491-92706-9.
13. Герберт Шилдт. C# 4.0: полное руководство = C# 4.0 The Complete Reference. — М.: «Вильямс», 2010. — С. 1056. — ISBN 978-5-8459-1684-6.

14. Кристиан Нейгел, Карли Уотсон и др. Visual C# 2010: полный курс = Beginning Microsoft Visual C# 2010. — М.: Диалектика, 2010. — ISBN 978-5-8459-1699-0.

15. Хокинг, Джозеф. Unity — в действии. Мультиплатформенная разработка на C# : [рус.]. — 2. — СПб : Питер, 2016. — 336 с. — ISBN 978-1617292323.

16. Торн, Алан. Искусство создания сценариев в Unity : [рус.]. — СПб : ДМК, 2016. — 362 с.

17. Горобець Ю., А. Кучко, І. Вавілов. Фрактальна геометрія в природничих науках / Ю. Горобець, А. Кучко, І. Вавілов // –Наукова думка. – 2008. – с. 232.

18. Трегубова І.А., К.О. Собко, Р.О. Гохман. Фрактальна графіка як сучасна технологія візуалізації / І.А. Трегубова, К.О. Собко, Р.О. Гохман. // Цифрові технології. – Україна, Одеса: ОНАТ ім О.С.Попова. – 2018. –№24. – с.111–117

19. Шляхтина С. Обзор решений для генерации изображений на основе фракталов и атракторов [Электронний ресурс]. – Режим доступу: https://compress.ru/article.aspx?id=21776

20. Zalevska, O., & Zakharkin , M. (2021). Disadvantages and advantages of using graph theory in video games. Modern Problems of Modeling, (20), 100-106. https://doi.org/10.33842/2313-125X/2021/20/100/10,

21. Vanin V., Zalevska O., Jablonskiy P. Застосування теорії графів для удосконалення та візуалізації алгоритму пошуку найкоротшого шляху в математичній моделі відео ігри //Прикладна геометрія та інженерна графіка. – 2020. – №. 97. – С. 23-28.

22. Buckley D., Unity vs. Unreal – Choosing a Game Engine – Режим доступу: https://gamedevacademy.org/unity-vs-unreal/

ДОДАТОК 1

Фрактали у відеоіграх

СПЕЦИФІКАЦІЯ

УКР.КПІім.ІгоряСікорського_ТЕФ_АПЕПС_ДА3229_19Б

Аркушів 1

Київ 2021

Позначення	Найменування	Примітки
Документація		
УКР.КПІім.ІгоряСікорського_ТЕФ_АПЕПС_ТР7209_21Б	Записка.docx	Текстова частина дипломної роботи
	Діаграми.vsdx	UML діграми
Компоненти		
УКР.КПІім.ІгоряСікорського_ТЕФ_АПЕПС_ТР7209_21Б	MazeController.cs	Модуль основного тіла програми
УКР.КПІім.ІгоряСікорського_ТЕФ_АПЕПС_ТР7209_21Б	MazeGenerator.cs	Виконання алгоритму генерації лабіринту
УКР.КПІім.ІгоряСікорського_ТЕФ_АПЕПС_ТР7209_21Б	PlayerController.cs	Персонаж
УКР.КПІім.ІгоряСікорського_ТЕФ_АПЕПС_ТР7209_21Б	RecursiveDivision.cs	Метод рекурсивного ділення
УКР.КПІім.ІгоряСікорського_ТЕФ_АПЕПС_ТР7209_21Б	SierpinskiCarpet.cs	Реалізація килиму Серпінського
УКР.КПІім.ІгоряСікорського_ТЕФ_АПЕПС_ТР7209_21Б	TreasureScript.cs	Управління текстурами
УКР.КПІім.ІгоряСікорського_ТЕФ_АПЕПС_ТР7209_21Б	FpsMovement.cs	Управління частотою кадрів
Позначення	Найменування	Примітки
Документація		

ДОДАТОК 2

Фрактали у відеоіграх

ТЕКСТ ПРОГРАМНОГО МОДУЛЮ

УКР.КПІім.ІгоряСікорського_ТЕФ_АПЕПС_ДА3229_19Б

Аркушів 19

Київ 2021

Файл TreasureScript

```
using UnityEngine;

public class TreasureScript : MonoBehaviour
{
    public GameObject mazeController;

    void OnTriggerEnter(Collider other)
    {
        var player = other.gameObject.GetComponent<PlayerController>();

        if (player is null) return;
        player.Score++;
        player.RemainingTime += 20;

        FindObjectOfType<MazeController>().GenerateTreasure();

        Destroy(gameObject);
    }
}
```

Файл FpsMovement

```
using UnityEngine;

[RequireComponent(typeof(CharacterController))]

public class FpsMovement : MonoBehaviour
{
```

```
[SerializeField] private Camera headCam;

public float speed = 6.0f;
public float gravity = -9.8f;

public float sensitivityHor = 9.0f;
public float sensitivityVert = 9.0f;

public float minimumVert = -45.0f;
public float maximumVert = 45.0f;

private float rotationVert = 0;

private CharacterController charController;

void Start()
{
   charController = GetComponent<CharacterController>();
}

void Update()
{
   MoveCharacter();
   RotateCharacter();
   RotateCamera();
}

private void MoveCharacter()
```

```
{
    float accelerate = Input.GetButton("Accelerate") ? 2 : 1;

    float deltaX = Input.GetAxis("Horizontal") * speed * accelerate;
    float deltaZ = Input.GetAxis("Vertical") * speed * accelerate;

    Vector3 movement = new Vector3(deltaX, 0, deltaZ);
    movement = Vector3.ClampMagnitude(movement, speed*accelerate);

    movement.y = gravity;
    movement *= Time.deltaTime;
    movement = transform.TransformDirection(movement);

    charController.Move(movement);
}

private void RotateCharacter()
{
    transform.Rotate(0, Input.GetAxis("Mouse X") * sensitivityHor, 0);
}

private void RotateCamera()
{
    rotationVert -= Input.GetAxis("Mouse Y") * sensitivityVert;
    rotationVert = Mathf.Clamp(rotationVert, minimumVert, maximumVert);

    headCam.transform.localEulerAngles = new Vector3(
        rotationVert, headCam.transform.localEulerAngles.y, 0
```

```
        );
    }
}
```

Файл MazeController

```
using UnityEngine;

public class MazeController : MonoBehaviour
{
    private MazeGenerator generator;

    public Material wallMaterial;
    public Material floorMaterial;

    public GameObject treasurePrefab;

    void Start()
    {
        var sier = Random.Range(0, 2) == 0;
        sier = true;
        generator = sier
            ? (MazeGenerator) new SierpinskiCarpet()
            : new RecursiveDivision();
        generator.Side = sier ? (int)Mathf.Pow(3, 4) : Random.Range(10, 15);

        generator.Parent = transform;
        generator.WallMaterial = wallMaterial;
        generator.FloorMaterial = floorMaterial;
```

```
        generator.TreasurePrefab = treasurePrefab;

        generator.CreateMaze();
        GenerateTreasure();
    }

    public void GenerateTreasure()
    {
        generator.GenerateTreasure();
    }
}
```

Файл MazeGenerator

```
using UnityEngine;

public abstract class MazeGenerator
{
    public int Side { get; set; }

    public Material WallMaterial { get; set; }
    public Material FloorMaterial { get; set; }
    public Transform Parent { get; set; }

    public GameObject TreasurePrefab { get; set; }

    public abstract void CreateMaze();
    public abstract void GenerateTreasure();
}
```

Файл PlayerController

```
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

[RequireComponent(typeof(FpsMovement))]
public class PlayerController : MonoBehaviour
{
    private FpsMovement player;
    public Text timeLabel;
    public Text scoreLabel;
    public GameObject endGameLabel;

    private int score;
    public int Score
    {
        get => score;
        set
        {
            score = value;
            scoreLabel.text = score.ToString();
        }
    }

    [SerializeField]
    private float remainingTime;
    public float RemainingTime
```

```
{
    get => remainingTime;
    set
    {
        remainingTime = value;
        timeLabel.text = ((int)Mathf.Floor(remainingTime)).ToString();
    }
}

// Start is called before the first frame update
void Start()
{
    player = gameObject.GetComponent<FpsMovement>();
    player.enabled = true;
}

// Update is called once per frame
void Update()
{
    if (!player.enabled)
    {
        if (Input.GetButtonDown("Submit"))
        {
            SceneManager.LoadScene(0);
        }
        return;
    }
```

```
        RemainingTime -= Time.deltaTime;

        if (RemainingTime <= 0)
        {
            endGameLabel.SetActive(true);
            timeLabel.text = "TIME UP";
            player.enabled = false;
        }
    }
}
```

Файл RecursiveDivision

```
using UnityEngine;

public class RecursiveDivision : MazeGenerator
{
    private int Width => Side;
    private int Height => Side;

    private const int S = 1;
    private const int E = 2;

    private const float WallHeight = 4f;
    private const float Scale = 3;

    private enum Orientation
    {
        Horizontal = 1,
```

```
    Vertical = 2
}

public override void CreateMaze()
{
    var board = new int[Height, Width];
    Divide(board, 0, 0, Width, Height, choose_orientation(Width, Height));
    display_maze(board);
}

public override void GenerateTreasure()
{
    int x = Random.Range(0, Width);
    int y = Random.Range(0, Height);

    var treasure = Object.Instantiate(TreasurePrefab);
    treasure.transform.position = new Vector3((x-0.5f)*Scale, -1, (y-0.5f)*Scale);
}

private static Orientation choose_orientation(int width, int height)
{
    if (width < height)
    {
        return Orientation.Horizontal;
    }

    if (height < width)
    {
```

```
            return Orientation.Vertical;
        }

        return Random.Range(0, 2) == 0
            ? Orientation.Horizontal
            : Orientation.Vertical;
    }

    private void Divide(int[,] board, int x, int y, int width, int height, Orientation orientation)
    {
        if (width < 2 || height < 2)
            return;

        bool horizontal = orientation == Orientation.Horizontal;

        // where will the wall be drawn from?
        int wx = x + (horizontal ? 0 : Random.Range(0, width - 2));
        int wy = y + (horizontal ? Random.Range(0, height - 2) : 0);

        // where will the passage through the wall exist?
        int px = wx + (horizontal ? Random.Range(0, width) : 0);
        int py = wy + (horizontal ? 0 : Random.Range(0, height));

        // what direction will the wall be drawn?
        int dx = horizontal ? 1 : 0;
        int dy = horizontal ? 0 : 1;
```

```
// how long will the wall be?
int length = horizontal ? width : height;

// what direction is perpendicular to the wall?
int dir = horizontal ? S : E;

for (int i = 0; i < length; i++)
{
   if (wx != px || wy != py)
   {
      board[wy, wx] |= dir;
   }

   wx += dx;
   wy += dy;
}

int nx = x;
int ny = y;

int w = horizontal ? width : wx - x + 1;
int h = horizontal ? wy - y + 1 : height;

Divide(board, nx, ny, w, h, choose_orientation(w, h));

nx = horizontal ? x : wx + 1;
ny = horizontal ? wy + 1 : y;
```

```
    w = horizontal ? width : x + width - wx - 1;
    h = horizontal ? y + height - wy - 1 : height;

    Divide(board, nx, ny, w, h, choose_orientation(w, h));
}

private void create_plane(Material material, Vector3 position, Vector3 scale)
{
    var plane = GameObject.CreatePrimitive(PrimitiveType.Cube);
    plane.GetComponent<MeshRenderer>().material = material;
    plane.transform.position = position;
    plane.transform.parent = Parent;
    plane.transform.localScale = scale;
}

private void display_maze(int[,] board)
{


    create_plane(WallMaterial,
        new Vector3(Scale * (Width / 2f - 1), 0, -Scale),
        new Vector3(Scale * Width, WallHeight, 0.1f));

    create_plane(WallMaterial,
        new Vector3(-Scale, 0, Scale * (Height / 2f - 1)),
        new Vector3(0.1f, WallHeight, Scale * Height));

    // ceiling
```

```
        create_plane(FloorMaterial,
            new Vector3(Scale * (Width / 2f - 1), WallHeight / 2, Scale * (Height / 2f -
1)),
            new Vector3(Scale * Width, 0.1f, Scale * Height));

        // floor
        create_plane(FloorMaterial,
            new Vector3(Scale * (Width / 2f - 1), -WallHeight / 2, Scale * (Height / 2f
- 1)),
            new Vector3(Scale * Width, 0.1f, Scale * Height));

        // generated walls
        for (int y = 0; y < board.GetLength(0); y++)
        {
            for (int x = 0; x < board.GetLength(1); x++)
            {
                var bottom = y + 1 >= board.GetLength(0);
                var south = (board[y, x] & S) != 0 || bottom;
                var south2 = x + 1 < board.GetLength(1) && (board[y, x + 1] & S) != 0
|| bottom;
                var east = (board[y, x] & E) != 0 || x + 1 >= board.GetLength(1);

                if (south)
                {
                    create_plane(WallMaterial,
                        new Vector3(Scale * (x - 0.5f), 0, Scale * y),
                        new Vector3(Scale, WallHeight, 0.1f));
                }
                if (east)
```

```
            {
                create_plane(WallMaterial,
                    new Vector3(Scale * x, 0, Scale * (y - 0.5f)),
                    new Vector3(0.1f, WallHeight, Scale));
            }
            else if (south && south2)
            {
                create_plane(WallMaterial,
                    new Vector3(Scale * (x - 0.5f), 0, Scale * y),
                    new Vector3(Scale, WallHeight, 0.1f));
            }

        }
    }
  }
}
```

Файл SierpinskiCarpet

```
using UnityEngine;

public class SierpinskiCarpet : MazeGenerator
{
    private bool[,] board;

    public override void CreateMaze()
    {
        board = new bool[Side, Side];
```

```
        GenerateFractal(0, 0, Side / 3);

        int scale = 2;

        for (int x = 0; x < Side; x++)
        {
            for (int y = 0; y < Side; y++)
            {
                if (!board[x, y] || (y-1 > 0 && board[x, y-1]) || (x-1 > 0 && board[x-1,
y]))
                    continue;

                int koef = 1;
                while (x < Side && y < Side && board[x+koef, y+koef])
                {
                    koef++;
                }

                if (koef > 1)
                {
                    var cube = GameObject.CreatePrimitive(PrimitiveType.Cube);
                    cube.transform.position = new Vector3((scale*x+koef), 0,
(scale*y+koef));
                    cube.transform.parent = Parent;
                    cube.transform.localScale = new Vector3(1.5f*koef, 3, 1.5f*koef);
                    cube.GetComponent<MeshRenderer>().material = WallMaterial;
                }
                else
```

```
            {
                var cube = GameObject.CreatePrimitive(PrimitiveType.Cube);
                cube.transform.position = new Vector3(scale*x+koef, 0,
scale*y+koef);
                cube.transform.parent = Parent;
                cube.transform.localScale = new Vector3(koef, 3, koef);
                cube.GetComponent<MeshRenderer>().material = WallMaterial;
            }
        }
    }

    // floor
    var plane = GameObject.CreatePrimitive(PrimitiveType.Cube);
    plane.transform.position = new Vector3(scale*Side / 2f, -1.5f, scale*Side /
2f);
    plane.transform.parent = Parent;
    plane.transform.localScale = new Vector3(scale*Side, 0.1f, scale*Side);
    plane.GetComponent<MeshRenderer>().material = FloorMaterial;

    // ceil
    plane = GameObject.CreatePrimitive(PrimitiveType.Cube);
    plane.transform.position = new Vector3(scale*Side / 2f, 1.5f, scale*Side / 2f);
    plane.transform.parent = Parent;
    plane.transform.localScale = new Vector3(scale*Side, 0.1f, scale*Side);
    plane.GetComponent<MeshRenderer>().material = FloorMaterial;

    // walls
    plane = GameObject.CreatePrimitive(PrimitiveType.Cube);
```

```
    plane.transform.position = new Vector3(scale*Side/2f, 0, 0);
    plane.transform.parent = Parent;
    plane.transform.localScale = new Vector3(scale*Side, 3, 0.1f);
    plane.GetComponent<MeshRenderer>().material = WallMaterial;

    plane = GameObject.CreatePrimitive(PrimitiveType.Cube);
    plane.transform.position = new Vector3(scale*Side/2f, 0, scale*Side);
    plane.transform.parent = Parent;
    plane.transform.localScale = new Vector3(scale*Side, 3, 0.1f);
    plane.GetComponent<MeshRenderer>().material = WallMaterial;

    plane = GameObject.CreatePrimitive(PrimitiveType.Cube);
    plane.transform.position = new Vector3(0, 0, scale*Side/2f);
    plane.transform.parent = Parent;
    plane.transform.localScale = new Vector3(0.1f, 3, scale*Side);
    plane.GetComponent<MeshRenderer>().material = WallMaterial;

    plane = GameObject.CreatePrimitive(PrimitiveType.Cube);
    plane.transform.position = new Vector3(scale*Side, 0, scale*Side/2f);
    plane.transform.parent = Parent;
    plane.transform.localScale = new Vector3(0.1f, 3, scale*Side);
    plane.GetComponent<MeshRenderer>().material = WallMaterial;
}

public override void GenerateTreasure()
{
    int x;
    int y;
```

```
    do
    {
        x = Random.Range(0, board.GetLength(0));
        y = Random.Range(0, board.GetLength(1));
    } while (board[x, y]);

    var treasure = Object.Instantiate(TreasurePrefab);
    treasure.transform.position = new Vector3(x, -1, y);
}

private void GenerateFractal(int startX, int startY, int blockSize)
{
    if (blockSize < 1)
        return;

    for (int x = startX + blockSize; x < startX + 2 * blockSize; x++)
    {
        for (int y = startY + blockSize; y < startY + 2 * blockSize; y++)
        {
            board[x, y] = true;
        }
    }

    for (int x = 0; x < 3; x++)
    {
        for (int y = 0; y < 3; y++)
        {
```

```
                int newBlockSize = blockSize / 3;
                GenerateFractal(startX + blockSize * x, startY + blockSize * y, new-
BlockSize);
            }
        }
    }
}
```

ДОДАТОК З

Фрактали у відеоіграх

ОПИС ПРОГРАМНОГО МОДУЛЮ

УКР.КПІім.ІгоряСікорського_ТЕФ_АПЕПС_ДА3229_19Б

Аркушів 5

Київ 2021

АНОТАЦІЯ

Додаток містить опис основного модуля відеогри типу «біг в лабіринті».

В додатку виконуються такі функції:

— Генерація випадкового фрактального лвбіринта;

— Виклик модулю управління персонажем;

— Пошук жовтих коробок за відведений час.

Гра реалізована засобами Unity та мова програмування C#.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку описано систему роботи основного модуля відеогри.

Модулі системи описані в ДОДАТКУ 2.

Додаток працює в операційних системах, таких як Windows7, Windows8, Windows10.

Компоненти необхідні для установки: відсутні.

Використана мова програмування — C#.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблена відеогра типу «біг у лабіринті» на основі генерації фрактальних лабіринтів методом рекурсивного ділення та побудови фрактала килим Серпінського.

Дана система може бути використана структурами, які займаються розробкою відеоіг.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Інформаційна система являє собою відеогру з простою та зручною системою керування гравцем, та не складною ігровою механікою пошуку коробок у лабіринті за відведений час.

Потрібно встановити всі програмні додатки гри та запустити програму. Після цього ви можете грати в гру, а по закінченню таймера вона почнеться знову

ВИКОРИСТОВУВАНІ ТЕХІЧНІ ЗАСОБИ

Компоненти необхідні для установки інформаційної системи: відсутні.

Використана мова програмування для інформаційної системи — C#.

Розроблена інформаційна система працює в операційних системах Windows7, Windows8, Windows10 та не потребує встановлення компонентів.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Для запуску необхідно запустити .exe файл і дочекатися запуску гри. Після цього ви можете керувати персонажем гравця стандартною комбінацією клавіш для руху «WASD», збирати жовті коробки заходячи персонажем на них. Після успішного підняття коробки, лічильник збільшиться на одиницю, а таймер на 20 секунд, а після завершення таймеру посеред екрану висвітиться повідомлення про завершення гри. При натисканні клавіши «Enter», гра почнеться з початку.

ДОДАТОК 4

АНАЛІЗ СФЕР ЗАСТОСУВАННЯ ФРАКТАЛІВ У ВІДЕОІГРАХ

ТЕКСТ ПУБЛІКАЦІЇ

УКР.КПІім.ІгоряСікорського_ТЕФ_АПЕПС_ДА3229_19Б

Аркушів 10

Київ 2021

УДК 514.18

АНАЛІЗ СФЕР ЗАСТОСУВАННЯ ФРАКТАЛІВ У ВІДЕОІГРАХ

DOI

Ладогубець Т. С. к.т.н.,

aladog@gmail.com, ORCID 0000-0001-8700-5477, H-index 1

Голова О.О., к.т.н.,

fire19@ukr.net, ORCID:0000-0002-4903-4450

Мірошніченко І.В.

Goodgod@ukr.net, ORCID 0000-0001-7383-8013

Паламар І.О.*,

palamarinna951@gmail.com, ORCID: 0000-0002-6184-1917

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського» (Київ, Україна)

З кожним днем відеоігри набувають все більшої популярності, зростає попит на нові креативні рішення як і у ігровому процесі, так і у графіці. Велику роль у вирішені даної проблеми відіграють фрактали. Вонизнайшли багато варіантів використання: від зображення дерев та об'єктіву 3D просторі, побудови ландшафтів карт до інтеграції їх у гемплей на більш глибшому рівні, як наприклад побудови основної концепції гри, коли однакові невеликі частини гри є частинами більшої, але все ще подібною наних по концепції. З цього постає необхідність у пошуку нових способів використання фрактальних алгоритмів для покращення графічного та концептуального рівня гри.

У даній роботі проведено аналіз способів використання фракталів для побудови ландшафтів карт та спрощення процесу реалізації

ландшафтного заповнення локацій; генерації складних лабіринтів; алгоритми процедурної генерації 3D лабіринтів відеогри на основі кросплатформленого ігрового двигуна; організацію взаємодії гравця з гроюза допомогою фракталів; можливість його використання у інших типах відеоігор. Наведено огляд ігрових застосунків в яких використовується фрактальна графіка, а саме: The Legend of Zelda, Link's Awakening, Свідок,Пробудження Лінка, I Love Hue.

Для спрощення використання фракталів при побудові лабіринтів наведено їх класифікацію в залежності від властивостей, якими вони володіють. До таких властивостей можна віднести розмірность, гіперрозмірність, топологію відтворення, тесселяцію, маршрутизацію, текстуру і пріоритет. Лабіринт може використовувати по одному елементу з кожного наведеного класу в будь-якому поєднанні. Використання фракталів в відеоіграх дозволяє значно зменшити об'ємоперативної пам'яті, необхідний для функціонування гри та збільшити її швидкодію за рахунок оптимізації алгоритму.

***Ключові слова**: фрактал, фрактальний дизайном гри, Fractal Design, Link's Awakening, фрактальна графіка, фрактальні лабіринти, фрактальнемоделювання.*

Постановка проблеми**.**

З кожним днем відеоігри набувають все більшої популярності, зростає й попит на нові креативні рішення як і у ігровому процесі, так і у графіці. Велику роль у вирішені даної проблеми грають фрактали. Вони знайшли багато варіантів використання: від зображення дерев та об'єктів у3D просторі, побудови ландшафтів карт до інтеграції їх у гемплей на біль глибшому рівні, як наприклад побудови основної концепції гри коли однакові невеликі частини гри є частинами більшої, але все ще подібною наних по концепції.

З цього постає необхідність у пошуку нових способів використання фрактальних алгоритмів для покращення вражень від процесу гри, як і у графічному, так і у концептуальному рівні.

Аналіз останніх досліджень і публікацій. В роботі [1] розглянуто створення гри за допомогою бібліотеки 2.2.1 SFML. SFML (Проста і Швидка Мультимедійна Бібліотека) є портативним і легким у використанні API для мультимедіа-програмування. SFML забезпечує використання 2D-графіки з апаратним прискорення OpenGL. SFML також надає різні модулі для полегшення програмування ігор і мультимедійних додатків[2]. В роботі [3]розглянуто побудову віртуальних алгоритмів для відображення ігрового світу. В

роботі [4] наведено графічні можливості для побудови зв'язку між ігровими персонажами. В роботі [5] розглянуті можливі варіанти удосконалення алгоритму пошуку мінімальної відстані до динамічної цілі.

Формулювання цілей статті. Розглянути існуючі підходи використання фракталів та фрактальної графіки при створенні відеогри. Надати рекомендації їх застосування в графіці гри та при взаємозв'язку між персонажами.

Основна частина. В даній роботі розглядаємо фрактал як самоподібну структуру, що підкорюється певним закономірностям. На основі вивчених фрактальних закономірностей, знайдених в природі, можливо створювати програми для їх імітації. Вивчаючи схеми розгалуження дерев, можна процедурно генерувати дерева програмним шляхом, а трохи змінюючи параметри цих фракталів, можливо створити нескінченну кількість унікальних, реалістичних дерев.

Фрактали також можна застосовувати в більших масштабах для створення ландшафтів. Багато реальних географічних об'єктів демонструютьфрактальну схожість: узбережжя, гори та річки. Фрактали використовують для генерації місцевостей автоматично при цьому отримуючи різніландшафтні зображення. Використання фрактальної залежності при побудові зображення об'єктів гри називають фрактальним дизайном гри. Подібно до того, як фрактал виглядає схожим при збільшенні та зменшеннімасштабу, деякі відеоігри роблять те саме - не на візуальному рівні, а на механічному. Наведемо приклади використання фракталів у відеоіграх.

Розглянемо гру Link's Awakening. У цій грі Лінк опиняється на загадковому острові, повному дивних мешканців, без видимого шляху. В процесі гри ви натикаєтесь на різні підземелля. Кожне підземелляпредставляє собою фрактальний лабіринт з перепонами [6].

Link's Awakening можна розглядати як фрактал глибиною в 3 шари. На зовнішньому шарі ви маєте сам острів Кохолінт, як більша загальна загадка, яку намагається розв'язати гравець. Трохи збільшивши масштаб, ви отримаєте самі підземелля, а на найменшому шарі у вас є окремі кімнати підземелля. Цей фрактальний візерунок природним чином випливає з принципу дизайну, який керується єдиною суттєвою механічною ідеєю. Беручи цю механічну ідею та застосовуючи її до декількох областей гри, виможете створити гру, яка відчуває згуртованість, не надто повторюючись [7].

Інший приклад гра-головоломка Свідок. Основна ідея Свідка полягає в тому, щоб розв'язати ряд головоломок з дуже подібними умовами - провести лінію через сітку від початкового положення до кінцевого. Фрактали

застосовуються при побудові та відображенні десятків унікальнихповоротів цієї простої механіки. Також застосовується цей прийом на декількох шарах для створення фрактального ефекту. На найнижчому рівніу вас є кожна окрема головоломка, яку потрібно вирішити, щоб активуватинаступну головоломку в послідовності. Зменште трохи, і ви отримаєте групиголоволомок з подібною механікою, згруповані в унікальні підобласті, причому кожна підобласть представляє більшу головоломку, яку потрібно вирішити, щоб активувати лазер. Завдяки способу подачі гри, кожна головоломка є частиною чогось більшого [8].

Розглянемо гру I Love Hue. Суть гри полягає у розташуванні гравцем кольорових блоків у відповідному порядку відносно їх відтінків. Хоча вона і має схожу структуру. Проте через надмірне зловживання фрактального підходу і відсутності різноманітності в рівнях, гра меншою мірою захоплює гравця [9].

Часто фрактали використовують для побудови лабіринтів у грі. В загальному лабіринти можна поділити на двовимірні, трьох вимірні, n-вимірні та переплетіння. На рисунку 1 наведено розподіл фрактальнихлабіринтів, що використовуються в відеоіграх. Фраткальні лабіринти засновуються на випадковому числу, яке відповідає за фрактальну гомотетію лабіринту.

Рис.1 Класифікація алгоритмів побудови лабіринту

Застосування 3D фракталів в віртуальній реальності продемонстровано у грі Marble Marcher це ігровий простір створено єдинималгоритмом. В наведеній грі всі об’єкти підпорядковуються математичним залежностям [9].

Фрактальна графіка позбавлена більшості недоліків растрової та векторної графіки. Наведемо таблицю недоліків та переваг використання фракталів у відеоіграх.

Таблиця 1.Недоліки та переваги використання фракталів в відеоіграх

Переваги використання фракталів в відеоіграх	**Недоліки використання фракталів в відеоіграх**
Не великий об’єм даних (зберігаються лише алгоритми)	Непередбачувана поведінка об’єкту при зміні параметра системи
Деталізація об’єкту	Складні математичні формули для відтворення віртуальних та реальних об’єктів
Можливість відтворення з частини об’єкту повністю весь об’єкт	Складні зображення перевантажуються ЦП та ОЗУ
Простота в модифікації (зміна лише одного параметру системи може повністю змінити об’єкт)	Не підтримка існуючих програмних розробок для створення фрактальної графіки різними операційними системи
Відсутність «пікселізації» зображення або об’єкту	Обмеженість початкових фігур об’єкту
Можливість відтворення реальних об’єктів	Недостатня кількість спеціалістів в області фрактальної графіки
Об’ємність зображення	

Висновок. Проведений аналіз існуючих відеоігр вказав на ряд недоліків та переваг при використання фракталів для моделювання ігрового процесу. Зважаючи на огляд ігрових застосунків можливо визначити сфери в яких застосування фракталів буде оптимальним. До таких сфер використання наделать ігрова графіка (як реальна, так і віртуальна), взаємозв’язок між персонажами, ігри в яких задіяно віртуальні лабіринти та такі стратегія якихполягає у «частинка подібна цілому». Використання фракталів розширює клас графічних явних та неявних об’єктів та функціональні можливості гри.

Список літератури

1. Горобець Ю., А. Кучко, І. Вавілов. Фрактальна геометрія в природничих науках / Ю. Горобець, А. Кучко, І. Вавілов // –Наукова думка. – 2008. – с. 232.

2. Трегубова І.А., К.О. Собко, Р.О. Гохман. Фрактальна графіка яксучасна технологія візуалізації / І.А. Трегубова, К.О. Собко, Р.О.Гохман. // Цифрові технології. – Україна, Одеса: ОНАТ ім О.С.Попова. – 2018. –№24. – с.111–117
3. Шляхтина С. Обзор решений для генерации изображений на основе фракталов и атракторов [Электронний ресурс]. – Режим доступу: https://compress.ru/article.aspx?id=21776
4. Zalevska, O., & Zakharkin , M. (2021). Disadvantages and advantages of using graph theory in video games. *Modern Problems of Modeling*, (20), 100-106. https://doi.org/10.33842/2313-125X/2021/20/100/10, .
5. Vanin V., Zalevska O., Jablonskiy P. Застосування теорії графів для удосконалення та візуалізації алгоритму пошуку найкоротшого шляху в математичній моделі відео ігри //Прикладна геометрія та інженерна графіка. – 2020. – №. 97. – С. 23-28.
6. Buckley D., Unity vs. Unreal – Choosing a Game Engine - Режимдоступу: https://gamedevacademy.org/unity-vs-unreal/
7. Dealessandri M., What is the best game engine: is Unreal Engine right for you? - Режим доступу: https://www.gamesindustry.biz/articles/2020-01-16- what-is-the-best-game-engine-is-unreal-engine-4-the-right-game-engine-foryou
8. . Perry D., David Perry on Game Design: A Brainstorming Toolbox Rusel – DeMaria , 2008 p. – 304 ст
9. Nystrom R., Game Programming Patterns – Kindle Edition, 2014p. –295ст

АНАЛИЗ СФЕРИ ПРИМЕНЕНИЯ ФРАКТАЛОВ В ВИДЕОИГРАХ

Ладогубець Т. С. Голова О.А., Мирошниченко И.В. Пономар И.А.

С каждым днем видеоигры преобретают всю большуюпопулярность, создает описание новых креативных решений как в игровомпроцессе, так и в графике. Большую роль в решении данной проблемы играют фракталы. Они знакомы со многими вариантами использования: спомощью изображения деревьев и объектов в 3D пространстве, построения ландшафтов карт к интеграции их в гемплей на более глубокомуровне, как, например, построены основные концепции игры, когданекоторые незначительные части игры являются частями большей, но всееще похожей на них по концепции. Таким

образом, необходимость поиска новых способов использования фрактальных алгоритмов для улучшения графического и концептуального уровня игры.

При данной работе проводится анализ возможностей использования фракталов для построения земельных участков карт и внедрение процесса реализации земельных участков; генерации сложных лабиринтов; алгоритмы процедурной генерации 3D лабиринтов видеоигры на основе кросплатформленого игрового двигателя; организация взаимодействия графиков с игрой с помощью фракталов; можно его использовать в другихтипах видеоигр. Дан обзор игровых приложений в которых используется фрактальная графика, а именно: The Legend of Zelda, Link's Awakening, свидетель, Пробуждение Линка, I Love Hue.

Для упрощения использования фракталов при построении лабиринтов проведена их классификация в зависимости от властных структур, которыми они пользуются. К таким органам власти можно применять размерность, гиперрозмирнисть, топологию воспроизведения, тесселяцию,маршрутизацию, текстуру и приоритет. Лабиринт можно использовать по одним элементом с каждого введенного класса в любом сочетании.

Использование фракталов в видеоиграх позволяет уменьшить объем оперативной памяти, необходимый для функционирования игры и увеличить ее скорость за счет оптимизации алгоритма.

Ключевые слова: фрактал, фрактальный дизайн игры, Fractal Design, Link's Awakening, фрактальная графика, фрактальные лабиринты, фрактальное моделирование.

ANALYSIS OF FIELDS OF APPLICATION OF FRACTALS IN VIDEO GAMES

Tetiana Ladogubets, Olga Golova, Ivan Miroshnichenko,

Inna Palamar

With each passing day, video games are becoming increasingly popular, creating a description of new creative solutions both in the gameplay and in

graphics. Fractals play an important role in solving this problem. They are familiar with many uses: by depicting trees and objects in 3D space, building map landscapes to integrate them into gameplay at a deeper level, such as building basic game concepts where some minor parts of the game are parts of a larger one. but still similar to them in concept. Thus, the need to find new ways to use fractal algorithms to improve the graphical and conceptual level of the game.

In this work, an analysis of the possibilities of using fractals for the construction of land maps and the introduction of the process of land sales; generation of complex labyrinths; algorithms for procedural generation of 3D video game mazes based on a cross-platform game engine; organization of interaction of graphics with the game with the help of fractals; you can use it in other types of video games. An overview of game applications that use fractal graphics, namely: The Legend of Zelda, Link's Awakening, Witness, Awakening of the Link, I Love Hue.

To simplify the use of fractals in the construction of labyrinths, their classification is carried out depending on the power structures they use. Dimensions, hyperdimensionality, playback topology, tessellation, routing, texture, and priority can be applied to such authorities. The maze can be used for one element from each entered class in any combination.

The use of fractals in video games allows you to reduce the amount of RAM required for the operation of the game and increase its speed by optimizing the algorithm.

Keywords: fractal, fractal game design, Fractal Design, Link's Awakening, fractal graphics, fractal mazes, fractal modeling.

References

1. Gorobets, Y., A.Kuchko, and I. Vavilov (2008). Fraktal′na heometriya v pryrodnychykh naukakh (Fractal geometry in the natural sciences). Naukova dumka, { in Ukrainian}
2. Tregubova, I.A., K.O. Sobko, and R.O. Gokhman. (2018)"Fraktal′na hrafika yak suchasna tekhnolohiya vizualizatsiyi(Fractal Graphics as modern imaging technology).", Digital Technology, Odessa, O.S.Popov ONAT, №24, pp.111–117.
3. Shlyachtina, S.,"Obzor resheniy dlya generatsii izobrazheniy na baze fraktalov i attraktorov (Review of solutions for image generation based on fractals and attractors)." compress, https://compress.ru/article.aspx?id=21776
4. Zalevska, O., & Zakharkin , M. (2021). Disadvantages and advantages of

using graph theory in video games. *Modern Problems of Modeling*, (20), 100-106. https://doi.org/10.33842/2313-125X/2021/20/100/10 { in Ukrainian}

5. Vanin V., Zalevska O., Jablonskiy P.(2020) Stagnation of the theory of graphs for a more detailed visualization of the algorithm of the best way in the mathematical model of video games // Applied Geometry and Engineering Graphics. -No. 97 .-- S. 23-28. { in Ukrainian}
6. Buckley D., Unity vs. Unreal – Choosing a Game Engine - Режимдоступу: https://gamedevacademy.org/unity-vs-unreal/
7. Dealessandri M., What is the best game engine: is Unreal Engine right for you? - Режим доступу: https://www.gamesindustry.biz/articles/2020-01-16- what-is-the-best-game-engine-is-unreal-engine-4-the-right-game-engine-foryou
8. Perry D. (2008)David Perry on Game Design: A Brainstorming Toolbox Rusel – DeMaria , – 304 pp
9. Nystrom R.(2014) Game Programming Patterns – Kindle Edition–295pp

ДОДАТОК 5

ФРАКТАЛИ У КОМП’ЮТЕРНИХ ІГРАХ

ТЕКСТ ПУБЛІКАЦІЇ

УКР.КПІім.ІгоряСікорського_ТЕФ_АПЕПС_ДА3229_19Б

Аркушів 2

Київ 2021

Ладогубець Т. С., к.т.н.
Голова О.О., к.т.н.,
Мірошніченко І.В.
Паламар І.О.
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»

ФРАКТАЛИ У КОМП'ЮТЕРНИХ ІГРАХ

Розробка гри базується на таких принципах, як ігрова концепція, використані технології та графічний дизайн. Взаємодія персонажа з віртуальним ігровим всесвітом визначається концепцією гри, ігровий рушій забезпечує технічну базу, а за візуалізацію відповідає дизайн. В кожному з трьох принципів можуть бути застосовані фрактали.

В відеоіграх фрактали застосовуються для: зображення дерев та об'єктів у 3D просторі, побудови ландшафтів карт, інтеграції їх у гемплей на більш глибшому рівні ніж простий алгоритм, побудови основної концепції гри. Використання фракталів у відеоіграх дозволяє значно зменшити об'єм оперативної пам'яті, необхідний для функціонування гри та збільшити її швидкодію за рахунок оптимізації алгоритму. В відеогрі можна виділити три основні способи використання фракталів, це комп'ютерна графіка, побудова навколишнього середовища та основна ідея геймплею. Комп'ютерна графіка забезпечує побудову складних об'єктів типу дерев, лісів, трави та інше. До побудови навколишнього середовища можна віднести побудову ландшафту локації, лабіринту, рельєфу місцевості, русла річок та доріг.

До основних переваг застосування фракталів в іграх можна віднести мінімальну об'ємність та легкість в генерації зображення. Аналіз показує, що фрактальна графіка хоч і містить деякі недоліки, але переваги є набагато значимі.

Мостовенко Ол-др В., к.т.н.

Київський національний університет будівництва і архітектури (Україна)