Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студента *Куйбіди Данила Віталійовича*
(ПІБ)

академічної групи *121М-22-2*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *«121 Інженерія програмного забезпечення »*
(назва освітньої програми)

на тему: *Розробка та дослідження ефективності застосування процедурної генерації рівнів ігрових застосунків*

_____ *Д.В. Куйбіда*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
розділів кваліфікаційної роботи				
спеціальний	*Доц. Приходченко С.Д.*			

Рецензент	*Проф. Корнієнко В.І.*				
Нормоконтролер	Доц. Гуліна І.Г.				

Дніпро
2023

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)

_____ М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » _____ 2023 Року

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

студенту *121м-22-2* *Куйбіда Данило Віталійович*
(група) (прізвище та ініціали)

Тема кваліфікаційної роботи *Розробка та дослідження ефективності застосування процедурної генерації рівнів ігрових застосунків*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 09.10.2023 р. № 1227-с

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єктом досліджень в кваліфікаційній роботі є "процедурна генерація рівнів для 2D-ігор". Саме цей процес є предметом вивчення та аналізу. У статті розглядаються різні аспекти та методи процедурної генерації рівнів, її переваги та недоліки, а також застосування в різних жанрах ігор, включаючи рогалики, шутери і стратегії.

Предметом дослідження в даній статті є "процедурна генерація рівнів для 2D-ігор". Предмет дослідження вказує на те, що ця тема є основним об'єктом аналізу і дослідження в рамках статті. У цьому випадку, велика увага надається самому процесу генерації рівнів в 2D-іграх, його методам, перевагам, недолікам та застосуванню в ігровій індустрії.

Мета НДР – дослідження впливу процедурної генерації на геймдизайн.

Вихідні дані для проведення роботи – доступ до літератури та джерел інформації.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Новизна запропонованих рішень полягає в інноваційному використання процедурної генерації. Дослідження може вказати на нові способи використання процедурної генерації в іграх, включаючи вдосконалення геймплею, графіки та інших аспектів гри.

Практична цінність полягає в внесенні змін та покращення. За результатами аналізу ви вносите зміни до алгоритмів та ігрового процесу, якщо це необхідно. Ви намагаєтеся покращити роботу процедурної генерації та геймплею на основі зібраних даних та відгуку.

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Результати досліджень мають бути подані у вигляді, що дозволяє побачити та оцінити безпосереднє використання запропонованих методів. В результаті роботи повинне бути розроблене програмне забезпечення для аналізу ефективності використання процедурної генерації в ігрових застосунках.

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець**
Аналіз теми та постановка задачі	12.09.2023-30.09.2023
Розробка процедурної генерації для рівнів в ігрових застосунків.	01.10.2023-31.10.2023
Використання програми та аналіз отриманих результатів	01.11.2023-12.12.2023

6 РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект від реалізації результатів роботи очікується постійне вдосконалення і підтримка гри. Процедурна генерація дозволяє створювати нові рівні та контент після виходу гри, що може підтримувати ігру актуальною та збільшувати продажі на протязі довшого часу.

Соціальний ефект від реалізації результатів роботи очікується збільшення ігрової різноманітності. Процедурна генерація може створювати ігри з великою різноманітністю, що привертає різні аудиторії. Гравці можуть знайти ігри, які відповідають їхнім вподобанням і стилю гри.

Завдання видав _____ *Приходченко С.Д.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Куйбіда Д.В.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *12.09.2023 р.*

Термін подання кваліфікаційної роботи до ЕК *12.12.2023*

РЕФЕРАТ

Пояснювальна записка: 72 с., 25 рис., 2 додатки, 42 джерела.

Об’єкт досліджень – створення та аналіз ефективності процедурної генерації рівнів в ігрових застосунків.

Предмет досліджень – процедурна генерація рівнів для 2D-ігор.

Мета роботи: дослідження впливу процедурної генерації на геймдизайн..

Методи дослідження. Аналіз готових процедурно генерованих рівнів для визначення того, як вони відрізняються від рівнів, створених вручну. Це включає в себе аналіз густини об'єктів, розміщення перешкод, розподіл ворогів, та інші аспекти геймдизайну.

Наукова новизна отриманих результатів полягає у створенні ігрового контенту, який адаптується під гравця в реальному часі. Наприклад, рівні можуть змінюватися відповідно до дій гравця, що створює більш динамічний та викликаючий геймплей.

Практична цінність роботи полягає у полягає в внесенні змін та покращення. За результатами аналізу ви вносите зміни до алгоритмів та ігрового процесу, якщо це необхідно. Ви намагаєтеся покращити роботу процедурної генерації та геймплею на основі зібраних даних та відгуку.

Область застосування. Процедурна генерація дозволяє створювати великі світи зі структурованими рівнями, підземеллями, містами, та місцями пригод. Гравці можуть досліджувати різні локації і зустрічати різні персонажі під час своєї подорожі.

Значення роботи та висновки. Використання процедурної генерації дозволяє створювати ігри, які можна грати знову і знову, оскільки кожне нове проходження буде унікальним. Це робить гру цікавою та викликаючою для гравців, що сприяє продовженню тривалості гри та залученню гравців на довший час.

Прогнозні припущення про розвиток досліджень. Прогнозується, що процедурна генерація рівнів залишатиметься популярним інструментом серед розробників ігор, особливо серед інді-розробників. Запит на ігри зі змінними рівнями та різноманітним контентом буде зростати.

ABSTRACT

Explanatory note: 72 pp., 25 fig., 2 app, 42 srcs.

The object of research is creation and analysis of the effectiveness of procedural event generation in game scenes.

The subject of research is procedural generation of levels for 2D games.

The purpose of the work: the study of the influence of procedural generation on game design.

Research methods. Analysis of ready-made procedurally generated levels to determine how they differ from manually created levels. This includes analysis of object density, placement of obstacles, distribution of enemies, and other aspects of game design.

The scientific novelty of the obtained results lies in the creation of game content that adapts to the player in real time. For example, levels can change according to the player's actions, which creates a more dynamic and challenging gameplay.

The practical value of the work lies in making changes and improvements. Based on the results of the analysis, you make changes to the algorithms and gameplay, if necessary. You try to improve procedural generation and gameplay based on collected data and feedback.

Field of application. Procedural generation allows you to create large worlds with structured levels, dungeons, cities, and adventure locations. Players can explore different locations and meet different characters during their journey.

Value of work and conclusions. Using procedural generation allows you to create games that can be played over and over again, as each new playthrough will be unique. This makes the game interesting and challenging for the players, which helps to extend the duration of the game and engage the players for a longer time.

Predictive assumptions about the development of research. Procedural level generation is predicted to remain a popular tool among game developers, especially among indie developers. The demand for games with variable levels and diverse content will grow.

ЗМІСТ

ВСТУП

Актуальність теми. Процедурна генерація рівнів в 2D-іграх має актуальність завдяки кільком факторам, які сприяють різноманітості та нескінченості контенту. Процедурна генерація дозволяє створювати ігровий контент без значного зростання витрат на розробку. Це надає ігровим студіям можливість розширити геймплей та подарувати гравцям більше можливостей і досвідів.

Зважаючи на перший та другий пункти щодо актуальності процедурної генерації рівнів в 2D-іграх, давайте розглянемо їх детальніше: Розширення геймдизайну: Процедурна генерація дозволяє ігровим розробникам створювати велику кількість різних рівнів чи контенту, використовуючи складові частини, алгоритми та параметризацію. Це допомагає розширити геймплей, додавати різноманітність та унікальність до ігрового світу. Ось деякі ключові аспекти цього пункту: Заощадження часу та ресурсів: Замість ручного створення кожного рівня або різних варіацій, розробники можуть створювати систему, яка генерує їх автоматично. Це зменшує витрати на розробку і дозволяє швидше додавати новий контент до гри. Більше можливостей для ігроків: Гравці отримують доступ до широкого спектру різних рівнів та завдань, що допомагає розширити їх ігровий досвід і стимулює більш глибоке вивчення гри. Нескінченість геймплею: Процедурна генерація дозволяє створювати рівні та контент у реальному чи псевдовипадковому режимі. Це допомагає досягти нескінченності геймплею в таких аспектах: Відсутність повторень: Гравці не будуть стикається з однотипними рівнями чи завданнями, оскільки кожен новий геймплей буде унікальним. Це забезпечує підвищену реиграбельність, оскільки гравці не нудьгують від повторних проходжень. Постійний виклик: На початковому етапі гра може бути досить легкою, але з розвитком гравця процедурно створювані рівні можуть надавати високу складність та виклик, що дозволяє грати новачкам

та досвідченим гравцям. Завдяки цим аспектам, процедурна генерація рівнів у 2D-іграх дозволяє зберігати гравців у грі на довгий час, надаючи їм можливість постійно відкривати нові геймплейні сценарії і отримувати задоволення від ігрового процесу, що робить її надзвичайно актуальною для розробників і гравців.

Об’єкт досліджень: мова представлення знань про шахту в аварійній ситуації.

Предмет досліджень: методи прийняття рішень по управлінню аварійними процесами (в умовах невизначеності) у вугільних шахтах.

Мета дослідження: Метою роботи є розробка нечіткого алгоритму керування вентиляцією на ділянці шахти.

Методи дослідження. При вирішенні поставлених задач було виконано аналіз та наукове узагальнення літературних джерел по вихідним посиланням досліджень, використовувались положення і алгоритми нечіткої логіки.

Новизна отриманих результатів полягає у використанні логіко-лінгвістичних моделей аварійних процесів в комплексі з нечіткою логікою для реалізації методів ситуаційного управління для прийняття рішень керівниками аварійних робіт при ліквідації наслідків аварій.

Практичне значення роботи полягає у розробленні математичних методів, алгоритмів та програмного забезпечення для прийняття рішень по управлінню аварійним розподілом повітря, що забезпечать підвищення ефективності прийняття рішень по управлінню аварійним розподілом повітря у вугільних шахтах і забезпечить високий рівень безпеки шахтарів, зменшення вірогідності помилки та покрашення швидкості надання кінцевого результату.

Особистий внесок автора полягає в розробці теоретичної частини магістерської роботи, в дослідженні і систематизації знань про існуючі методики, розробці методів досліджень і технології реалізації, в оцінці отриманих результатів.

Структура та обсяг дипломної роботи. Робота складається з вступу, чотирьох розділів і висновків. Містить 69 сторінок друкованого тексту, в тому числі 57 сторінок тексту основної частини, 25 рисунків, перелік використаних джерел, 2 додатки на 8 сторінках.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАВДАННЯ

1.1. Аналіз предметної області

Зростання популярності процедурної генерації в ігровій індустрії - це важливий феномен, який можна пояснити з декількох різних перспектив і аспектів. Процедурна генерація рівнів у 2D-іграх стає все більш важливою і актуальною складовою сучасного гейм дизайну та індустрії в цілому, і це стосується як незалежних розробників, так і великих видавництв. Спершу, варто відзначити, що геймінг стає все більш інтерактивним та споживає більше контенту, ніж будь-коли раніше. Гравці шукають нові враження, виклики та розваги, і саме процедурна генерація дозволяє забезпечити це. Замість статичних та передбачуваних рівнів, гравцям надається можливість досліджувати віртуальні світи, які створюються на льоту і мають нескінченну кількість варіантів. Це збільшує тривалість гри та робить її більш захопливою, оскільки гравці ніколи не знають, що чекає їх у наступному рівні. Друга перспектива полягає в ефективності та ресурсозбереженні для розробників. Створення рівнів вручну може бути дуже витратним за часом та коштами. Процедурна генерація дозволяє автоматизувати процес створення рівнів, зменшуючи ручну працю та сприяючи прискоренню розробки гри. Це особливо корисно для невеликих команд розробників та інді-геймдеву, які мають обмежені ресурси, але все одно бажають створити ігри високої якості. Третя перспектива пов'язана із змінами у смаках гравців. Сучасні гравці шукають глибокий та різноманітний ігровий досвід. Вони хочуть бути здивовані, і процедурна генерація допомагає задовольнити цю потребу. Замість передбачуваних сценаріїв гравцям пропонується нескінченна комбінація різних викликів та завдань, що створює новий рівень інтриги і цікавості. В заключення, зростання популярності процедурної генерації рівнів у 2D-іграх - це відповідь на зростаючий попит на

різноманітність, нескінченність та відкриття для гравців, а також на потребу розробників в ефективних методах створення контенту. Цей тренд відкриває нові можливості для гейм дизайнерів та розробників ігор, що сприяє подальшому розвитку індустрії та підвищує задоволення гравців від ігрового досвіду.

1.2. Огляд існуючих рішень

Огляд літератури та існуючих рішень для процедурної генерації рівнів у 2D-іграх з метою створення різноманітного та нескінченного контенту є важливим етапом у розумінні цієї предметної області.

Тайлова генерація є популярним методом для створення 2D-рівнів у відеоіграх. Вона базується на використанні тайлів, які є невеликими графічними блоками, що представляють частини ігрового світу (наприклад, землю, стіни, воду, дороги тощо). Рівень створюється шляхом розміщення цих тайлів у випадковому порядку або за певними правилами. Переваги тайлової генерації включають в себе простоту та швидкість розробки рівнів. Розробники можуть створювати набір тайлів та визначити правила їх розміщення, а генераційний алгоритм розміщує їх, створюючи рівень. Цей підхід дозволяє легко створювати різні комбінації рівнів, забезпечуючи різноманітність.

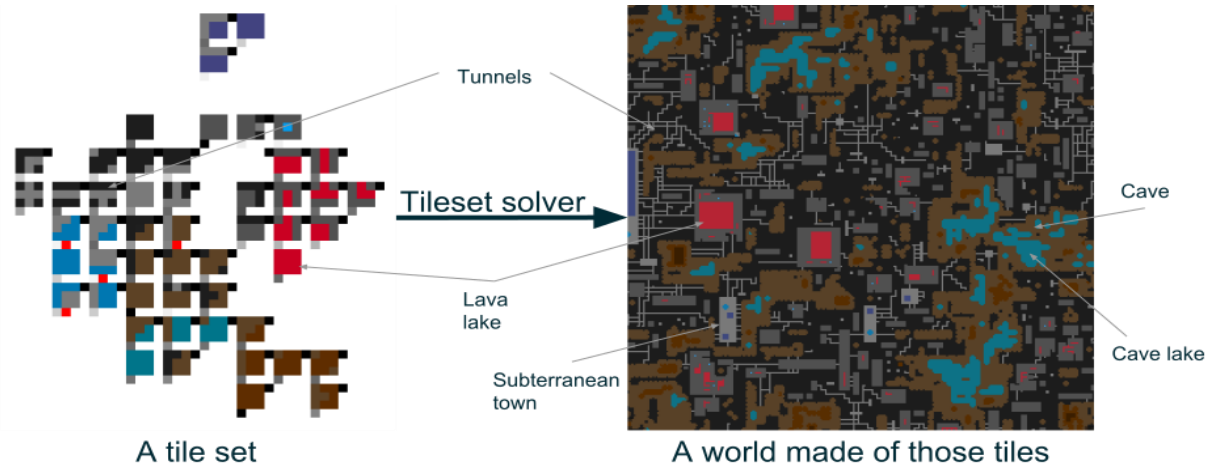

Рис. 1.1. Приклад тайлів

Принцип "Схожості та Різноманітності (Similarity and Variability Principle), який важливий для забезпечення рівноваги між схожістю та різноманітністю в процедурно генерованих рівнях. Він передбачає використання елементів схожості (наприклад, стандартних блоків, стін, доріг тощо), що повторюються на різних рівнях для створення знайомого середовища. Одночасно, цей принцип включає елементи різноманітності, які додають унікальність до кожного рівня (наприклад, різні декорації, враги, або сценарії). Збалансований підхід між схожістю та різноманітістю допомагає гравцям відчувати стабільність та розмаїтість у грі. Схожість створює відчуття об'єднаності в ігровому світі, а різноманітість робить гру цікавішою та викликаючою.

Генетичні алгоритми є методом, що моделює еволюційні процеси для створення рівнів. Розробники створюють початковий набір рівнів, які можуть бути представлені графічно або у вигляді даних, і визначають критерії, за якими ці рівні оцінюються. Потім алгоритм починає "мутувати" та комбінувати рівні, відбираючи найкращі варіанти на основі встановлених критеріїв. Цей процес повторюється декілька разів, створюючи нові рівні на кожній ітерації.

Рис. 1.2. Схематичний генетичний алгоритм

1.3 Постановка задачі

Постановка задачі для процедурної генерації рівнів в 2D-іграх з метою забезпечення різноманітності та нескінченності контенту - це важливий крок у розробці ігрового досвіду, який дозволяє гравцям отримувати нові та цікаві виклики кожного разу, коли вони грають. Ось загальна постановка завдання для цього:

Мета: Створити систему процедурної генерації рівнів для 2D-ігор, яка забезпечить різноманітність та нескінченність контенту, з метою поліпшення геймплею та реіграбельності.

Завдання: Створення базових блоків та об'єктів: Спроектувати базові блоки (тайли) та ігрові об'єкти (перешкоди, вороги, предмети) для використання в процедурній генерації. Це включає створення графіки, яка відповідає концепції гри. Визначення правил генерації: Розробити правила та алгоритми для розміщення блоків та об'єктів на рівнях. Ці правила можуть включати обмеження, такі як заборони на розміщення блоків у певних зонах, створення лабіринтів, або розміщення об'єктів на основі певних критеріїв. Варіабельність у рівнях: Впровадити елементи випадковості в генерацію, щоб забезпечити різноманітність. Це може включати в себе випадковий вибір тайлів, зміну розташування об'єктів, та інші параметри, які змінюються кожного разу, коли гравець запускає рівень. Баланс: Забезпечити баланс між спадковістю та геймплеєм. Розробити механізми для перевірки складності та грати, щоб гарантувати, що гравці отримують виклики, але не зустрічають надто важких або надто легких рівнів. Оптимізація: Враховуючи, що генерація рівнів може бути ресурсомісткою операцією, розробити методи оптимізації для прискорення процесу генерації та зменшення навантаження на систему.

1.4. Висновки з першого розділу

Процедурна генерація рівнів для 2D ігор стала важливим інструментом в геймдизайні та розробці ігор. Ця технологія дозволяє створювати різноманітні, унікальні та нескінченні ігрові світи, покращуючи якість геймплею та забезпечуючи більшу реіграбельність. Однак, важливо враховувати, що процедурна генерація не є панацеєю і вимагає грамотного підходу та розуміння геймдизайну. Застосування процедурної генерації в різних жанрах ігор, включаючи рольові ігри, платформери та головоломки, дозволяє розробникам створювати ігри, які завжди залишаються свіжими та цікавими для гравців. Це робить геймінг

більш доступним і різноманітним, і при цьому зберігає високий рівень якості. Прогнозуються подальші дослідження та розвиток процедурної генерації рівнів, включаючи вдосконалення алгоритмів та зростання їх інтелектуальної складності. Зростання популярності і використання в іграх з відкритим світом також варто очікувати. Загалом, процедурна генерація рівнів є важливою складовою сучасної індустрії ігор та має великий потенціал для подальшого розвитку і вдосконалення ігрового досвіду.

РОЗДІЛ 2

МЕТОДИ ТА ІНСТРУМЕНТИ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ РІВНІВ У ІГРОВИХ ЗАСТОСУНКАХ

2.1 Техніки та Алгоритми Процедурної Генерації

2.1.1 Генетичне програмування у генерації рівнів

Генетичне програмування є одним із методів процедурної генерації рівнів, який базується на імітації еволюційних процесів. У цьому підпункті детально описуються принципи роботи генетичного програмування для створення рівнів у іграх. Генетичне програмування (ГП) - це метод, який моделює природний процес еволюції для створення оптимальних рішень у просторі можливих варіантів. У вигляді алгоритму ГП використовується для процедурної генерації рівнів у ігрових застосунках.

Основна ідея полягає у тому, щоб створити початковий набір рівнів (або геномів), які представлені у вигляді послідовностей параметрів, наприклад, локацій об'єктів, типів противників, складності ландшафту тощо. Потім використовуються еволюційні процеси - селекція, схрещування, мутація - для того, щоб змінювати ці геноми таким чином, щоб отримати нові варіації рівнів.

Селекція в процедурній генерації рівнів - це процес відбору та використання створених або згенерованих об'єктів, елементів або рівнів у відповідь на певні критерії чи правила. У контексті генерації ігрових рівнів це важлива частина процесу створення, де обираються найкращі згенеровані елементи для побудови фінального рівня. Процедурна генерація рівнів використовує алгоритми та правила для створення ігрових середовищ без прямого втручання розробників. Саме в цьому контексті селекція стає важливою - вона відбирає найкращі чи найбільш підходящі елементи для створення рівня, відповідно до заданих критеріїв. Наприклад, враховуючи складність, рівень

випадковості, геймплейні аспекти або структуру рівня. Цей процес може включати оцінювання різних характеристик частин рівнів (таких як важкість, рівень ризику, інтересність тощо) і обрання тих, що найкраще відповідають бажаному результату. Селекція може базуватися на різних алгоритмах, від простих до складних, враховуючи ігрову механіку, побудову рівня та різноманітність геймплею. Це важлива частина процесу, яка дозволяє генеративним алгоритмам створювати цікаві та відмінні рівні, відповідні стилю ігри та вимогам гравців.

Схрещування - це ключовий етап у генетичних алгоритмах, який передбачає комбінування генетичного матеріалу двох батьківських об'єктів для створення нащадка з властивостями обох батьків. У процесі генерації рівнів у відеоіграх, це може застосовуватися для створення нових рівнів на основі характеристик чи параметрів існуючих рівнів. У контексті генетичних алгоритмів схрещування використовує різні методи комбінації генетичного матеріалу батьків, такі як одноточкове схрещування (застосування точки перетину для обміну частинами генетичної послідовності), двоточкове схрещування (використання двох точок перетину), уніформне схрещування (де кожен ген вибирається з обох батьків з певною ймовірністю), інтервальне схрещування та інші. У випадку генерації ігрових рівнів, після вибору кращих або підходящих елементів через селекцію, може використовуватися схрещування для комбінування їх характеристик або параметрів з метою створення нових, унікальних рівнів. Наприклад, якщо один рівень має високий рівень складності, а інший - цікаві пейзажі, схрещування може допомогти поєднати ці властивості у новому рівні. Схрещування в генетичних алгоритмах дозволяє створювати нові комбінації властивостей з вихідних об'єктів, що може призвести до покращення якості згенерованих рівнів та сприяти більшій різноманітності та цікавості в ігровому процесі.

Мутація - це процес в генетичних алгоритмах, що передбачає випадкове внесення змін у генетичний матеріал, що представляє індивіда. У контексті процедурної генерації рівнів в іграх, мутація може бути використана для випадкової зміни частин створених рівнів з метою отримання нових, унікальних конфігурацій. Мутація дозволяє вносити випадкові зміни в генетичний код, що може включати зміну параметрів, додавання або видалення частин елементів рівня, зміну положення об'єктів, зміну їх властивостей та багато іншого. Наприклад, мутація може змінити розташування ворогів, змінити розміри об'єктів або навіть змінити поведінку деяких елементів рівня. Цей процес важливий, оскільки він додає випадковість та різноманіття до згенерованих об'єктів або рівнів.

Мутація допомагає уникнути стагнації в алгоритмі генерації, дозволяючи отримувати нові комбінації характеристик та створювати більш різноманітні рівні. Проте, важливо збалансувати мутацію таким чином, щоб вона не вела до занадто випадкових або неігрових результатів. Такий баланс дозволить зберігати якість та цікавість згенерованих рівнів, уникнувши втрати ігрової цінності через надмірну випадковість.

Цей цикл селекції, схрещування і мутації повторюється кілька разів до досягнення оптимальних чи прийнятних за якістю рівнів. Генетичне програмування дозволяє автоматизувати процес створення рівнів, забезпечуючи велику різноманітність та експериментальний підхід до генерації ігрових рівнів, що може призвести до створення цікавих та унікальних геймплейних сценаріїв.

2.1.2 Алгоритми зграї частинок в генерації ігрових рівнів

Алгоритми зграї частинок (Particle Swarm Optimization, надалі - PSO) - це метод оптимізації, який моделює поведінку зграї частинок у просторі шукаючи оптимальне рішення для задачі. У контексті генерації ігрових рівнів, PSO може

використовуватися для створення оптимальних параметрів рівнів, таких як розташування об'єктів, характеристики противників, генерація територій тощо.

Основна ідея полягає в тому, щоб представити кожен потенційний рівень як частинку в просторі параметрів. Ці частинки рухаються у просторі пошуку рішення, оптимізуючи свої параметри на основі критерію якості (фітнес-функції), який оцінює, наскільки добре кожен рівень відповідає заданим критеріям.

Кожна частинка в зграї має свої власні параметри, такі як позиція та швидкість. Ініціалізація полягає у заданні початкових значень цих параметрів для кожної частинки. Це можуть бути випадкові значення, що лежать у межах допустимого діапазону, або обрані з якоїсь певної конфігурації. Крім того, кожна частинка має певну "пам'ять", де зберігається найкраще рішення, яке вона знайшла до того часу. Ця пам'ять допомагає керувати рухом кожної частинки у просторі пошуку, спрямовуючи її в напрямку більш перспективних рішень. Коли зграя частинок рухається у просторі пошуку, вона взаємодіє між собою. Частинки обмінюються інформацією про свої власні результати та знайдені найкращі рішення. Це взаємодія допомагає зграйові колективно рухатися в напрямку кращих рішень.

Один із ключових аспектів ініціалізації зграї полягає у встановленні початкових умов, які стимулюватимуть частинки до ефективного пошуку оптимального рішення. Це може включати розподіл частинок по простору пошуку таким чином, щоб покрити його рівномірно або зосередити у деяких областях, залежно від особливостей задачі.

У процесі (PSO) після початкової ініціалізації кожна частинка починає рух у просторі параметрів. Цей рух визначається її поточною позицією і швидкістю. Під час оновлення позицій та швидкостей частинок, вони пристосовуються з урахуванням кращих рішень, які вони знайшли самостійно та через інформаційний обмін з іншими частинками. Кожна частинка в алгоритмі PSO

має свою власну швидкість і позицію у просторі параметрів. Ці значення оновлюються на кожній ітерації алгоритму відповідно до формул, які враховують їхні попередні значення та інформацію від кращих рішень, знайдених зграєю. Оновлення швидкості кожної частинки зазвичай включає в себе комбінацію двох компонент: попередньої швидкості частинки та двох векторів. Перший вектор впливає на швидкість на основі попередньої швидкості та вектора між поточною позицією частинки та кращим знайденим рішенням нею самою (локальне найкраще рішення). Другий вектор враховує глобально краще знайдене рішення з усієї зграї. Ці два вектори впливають на швидкість з певними ваговими коефіцієнтами, що регулюють внесок кожного з них. Після оновлення швидкості частинки, обчислюється її нова позиція у просторі параметрів. Ця нова позиція визначається, використовуючи попередню позицію та оновлену швидкість. Таким чином, кожна частинка здійснює рух у просторі параметрів в напрямку, що відображає її пошук оптимального рішення. Оновлення позицій та швидкостей частинок в алгоритмі PSO відбувається на кожній ітерації, доки не буде досягнуто заданої умови зупинки (наприклад, досягнення максимальної кількості ітерацій чи заданої точності рішення).

Функція оцінки приймає на вхід значення параметрів, які характеризують певну позицію частинки, та повертає числове значення, що відображає якість цієї позиції. Чим менше або більше це значення (залежно від задачі: мінімізація чи максимізація), тим краще чи гірше вважається відповідна позиція. Оцінка кращої позиції відбувається порівнянням значення функції оцінки на поточній позиції частинки з її попередньо збереженою "пам'яттю" про найкращу знайдену раніше позицію. Якщо поточне значення функції оцінки краще, частинка оновлює свою "пам'ять" на поточну позицію, вважаючи її новою найкращою. Крім того, в PSO існує поняття глобальної кращої позиції для всієї зграї. Це позиція, яка визначається найкращим значенням функції оцінки серед усіх частинок зграї. Ця

позиція також зберігається та оновлюється протягом роботи алгоритму при знаходженні кращих рішень.

Таким чином, оцінка кращої позиції у PSO здійснюється порівнянням значень функції оцінки на поточній позиції кожної частинки з її попередньо збереженими кращими результатами (локально та глобально). Цей механізм дозволяє зграї здійснювати пошук в напрямку оптимальних рішень у просторі параметрів задачі.

Коли частинки здійснюють рух у просторі параметрів у процесі алгоритму зграї частинок, корекція цього руху відбувається з урахуванням декількох факторів. Кожна частинка в PSO має свою власну швидкість та позицію, які вони оновлюють на кожній ітерації. Однак важливо контролювати швидкість та рух частинок, щоб уникнути занадто швидкого або хаотичного руху, який може ускладнити процес знаходження оптимального рішення. Корекція руху частинок полягає у регулюванні їхньої швидкості та напрямку руху з метою кращого вирішення задачі оптимізації. Це зазвичай відбувається за допомогою встановлення максимальної швидкості для кожної частинки. Якщо поточна швидкість перевищує максимальне значення, вона може бути скоригована чи обмежена до заданого максимуму. Крім того, рух частинок може бути скоригований залежно від їхньої позиції в просторі параметрів.

Наприклад, можуть застосовуватися методи, що обмежують частинки у визначеній області пошуку, щоб уникнути виходу за межі допустимого простору параметрів. Це сприяє збереженню частинок у рамках області, де шукається оптимальне рішення.

У контексті алгоритму зграї частинок (PSO), критерій зупинки визначає умову, яка припиняє подальшу роботу алгоритму. Цей критерій служить ознакою завершення процесу оптимізації та вказує на досягнення потрібної умови для завершення роботи алгоритму. Зазвичай критерії зупинки можуть бути різними, вони встановлюються залежно від конкретної задачі або цілей

оптимізації. Наприклад, одним з критеріїв зупинки може бути досягнення максимальної кількості ітерацій, яку визначають перед запуском алгоритму. Іншим критерієм може бути досягнення заданої точності рішення: якщо результат оптимізації став задовільним для поточних потреб, алгоритм може завершити свою роботу. Також критерієм зупинки може стати досягнення певного значення функції оцінки, коли вона досягає певної межі або вже не покращується з достатньою швидкістю.

Використання PSO у генерації ігрових рівнів дозволяє швидко орієнтуватися у просторі параметрів та знаходити оптимальні комбінації, які відповідають заданим критеріям. Цей метод може бути корисним у випадках, коли прості еволюційні алгоритми не ефективні через складність функції оцінки або великий простір параметрів.

2.1.3 Генетичні Алгоритми у Процесі Генерації Рівнів

Генетичні алгоритми (ГА) - це еволюційний підхід до оптимізації, який моделює природний відбір та генетичні механізми для пошуку оптимальних рішень у просторі можливих варіантів. У контексті генерації ігрових рівнів, ГА може бути використаний для створення різноманітних та цікавих ігрових сценаріїв.

Основні компоненти генетичних алгоритмів включають: популяція, функція придатності, селекція, схрещування, мутація.

Алгоритм процедурної генерації популяцій використовується для створення групи об'єктів, персонажів чи сутностей у віртуальному середовищі. Це може бути популяція мешканців міста в ігровому світі, група тварин у віртуальній екосистемі або навіть армія ворогів у відеоіграх. Алгоритм процедурної генерації популяцій може використовувати різні підходи. Наприклад, може бути використана генетична або еволюційна модель, де група

об'єктів створюється шляхом комбінування характеристик, спадковості та зміни параметрів з часом. Це може наслідувати природні принципи еволюції, де кращі характеристики передаються наступним поколінням об'єктів у популяції. Інші підходи можуть використовувати випадкову генерацію або комбінування різноманітних параметрів для створення різноманітності в популяції. Зазвичай враховуються певні обмеження чи правила, щоб забезпечити баланс та адекватність створених об'єктів. Мета алгоритму генерації популяцій - створити групу об'єктів, яка буде виглядати реалістично, різноманітно та цікаво в контексті віртуального середовища. Застосування таких алгоритмів може допомогти розширити можливості та реалістичність віртуальних світів у відеоіграх, симуляціях чи інших програмних середовищах.

Функція придатності - це ключовий елемент багатьох алгоритмів оптимізації, включаючи алгоритми процедурної генерації. Ця функція визначає, наскільки добре вирішення або створений об'єкт відповідає бажаному критерію чи цілі. У контексті процедурної генерації, функція придатності оцінює якість згенерованих об'єктів. Наприклад, якщо ми генеруємо персонажів для ігри, функція придатності може враховувати такі параметри як швидкість, сила, здоров'я персонажа та його можливості адаптуватися до різних умов у грі. Для генерації рівнів у відеоіграх, функція придатності може оцінювати складність рівня, наявність цікавих елементів та готовність рівня до геймплею. Суть функції придатності полягає в тому, щоб мати чіткий критерій для оцінки створеного контенту та визначення того, наскільки він відповідає очікуванням або критеріям якості. Це допомагає алгоритмам процедурної генерації орієнтуватися в просторі можливих варіантів і знаходити ті, які найкраще відповідають поставленим завданням чи вимогам.

Селекція є ключовим етапом в алгоритмах оптимізації та еволюційних алгоритмах, таких як генетичні алгоритми. Це процес вибору найкращих рішень або об'єктів з популяції для подальшого сформування нових поколінь. У

контексті процедурної генерації, селекція може застосовуватися до об'єктів, що були згенеровані алгоритмами, для виділення найкращих з них для наступного етапу. Цей процес може використовувати функцію придатності для оцінки кожного об'єкта, і обирати ті, які найкраще відповідають визначеним критеріям. Існує декілька методів селекції, таких як турнірний відбір, рулетковий відбір, відбір за рангом тощо. Кожен з цих методів може мати свої переваги в різних ситуаціях та задачах. Селекція допомагає зберігати найкращі рішення або об'єкти в популяції, що сприяє покращенню якості згенерованого контенту з покоління в покоління. Це сприяє ефективнішому пошуку оптимальних рішень або об'єктів у просторі можливих варіантів.

Схрещування (або кросовер) в алгоритмах процедурної генерації є процесом створення нових об'єктів чи рішень шляхом комбінації характеристик або параметрів двох або більше об'єктів з попередніх поколінь. У генетичних алгоритмах, схрещування відбувається після вибору кращих рішень (селекції). Два обрані об'єкти (батьки) об'єднуються для створення нових об'єктів (нащадки) шляхом комбінування їхніх генетичних складових. Наприклад, у випадку, коли ми генеруємо персонажів у відеоіграх, схрещування може включати комбінування характеристик (наприклад, властивостей персонажів) батьків для створення нового персонажа з комбінованими атрибутами. Цей процес може відбуватися по-різному в залежності від конкретної задачі та алгоритму. Наприклад, в алгоритмах, що використовуються для генерації ігрових рівнів, схрещування може включати комбінування різних частин різних рівнів для створення нового рівня з елементами попередніх. Схрещування дозволяє зберегти та поєднати корисні характеристики кращих об'єктів для створення нових, що може призвести до вдосконалення об'єктів у кожному поколінні або ітерації алгоритму.

Мутація - це процес внесення випадкових змін у генетичну структуру об'єктів або рішень під час алгоритмічної обробки, такої як процедурна

генерація. Це важливий етап у еволюційних алгоритмах, оскільки дозволяє внести різноманітність і нововведення в популяцію. У контексті процедурної генерації, мутація може випадково змінювати параметри чи властивості об'єктів певним чином. Наприклад, в генерації персонажів для відеоігор, мутація може вплинути на властивості, які визначають характеристики персонажа (такі як швидкість, сила, витривалість) і змінювати їх на випадкові значення. Мутація дозволяє алгоритмам процедурної генерації досліджувати нові області в просторі можливих рішень та експериментувати з різноманітністю варіантів. Це допомагає уникнути застрягання на певних рішеннях та сприяє пошуку більш оптимальних або цікавих об'єктів. Також мутація може допомогти у виявленні нових, непередбачуваних рішень, які можуть бути корисними у віртуальному середовищі.

Ці етапи повторюються кілька разів (поколінь) до досягнення бажаного рівня якості або збіжності до оптимального рішення. Генетичні алгоритми можуть бути ефективними для генерації рівнів у іграх через їхню здатність до еволюції та адаптації до різноманітних критеріїв, що дозволяє створювати унікальні та цікаві ігрові сценарії, враховуючи велику кількість параметрів та умов для генерації рівнів.

2.2 Інструментальні Засоби для Процедурної Генерації Рівнів

2.2.1 Фреймворки та Ігрові Двигуни

Фреймворк це програмне середовище, яке спрощує та прискорює створення програмного забезпечення. За використання фреймворків ви пишете лише код, який реалізує логіку, специфічну для вашого продукту. Існує багато таких фреймворків, наприклад як Unity. Цей популярний фреймворк має

широкий спектр можливостей для розробки ігор. Включає у себе різноманітні інструменти, такі як Unity Procedural Generation Toolkit, які допомагають розробникам імплементувати процедурну генерацію рівнів. Цей фреймворк підтримує різні мови програмування, включаючи C#.

Ігрові двигуни, які підтримують процедурну генерацію рівнів, розширюють можливості розробників у створенні унікальних ігрових світів. Unreal Engine, Unity, та Godot - це лише деякі з популярних двигунів, які дозволяють використовувати процедурну генерацію. Унікальність кожного з цих двигунів полягає в тому, як вони підтримують та реалізовують процес процедурної генерації. Наприклад, Unreal Engine має вбудований набір інструментів, таких як Blueprints або C++, що дають можливість створення складних систем генерації рівнів безпосередньо в середовищі двигуна. Unity зі своїм Asset Store надає розробникам доступ до різноманітних плагінів та ресурсів, які полегшують і прискорюють процес процедурної генерації. Це можуть бути різні скрипти, пакети або готові модулі, які допомагають автоматизувати створення різноманітних ігрових об'єктів. Godot, з своєю власною системою скриптування, надає гнучкість та можливості для реалізації процедурної генерації. Розробники можуть використовувати GDScript або C# для створення власних алгоритмів, що генерують рівні, об'єкти або ландшафти у грі. Ці ігрові двигуни надають розробникам можливість експериментувати з процедурною генерацією та створювати унікальні ігрові світи без значного обсягу ручної роботи. Вони відкривають двері до створення непередбачуваних і захоплюючих ігрових досвідів, де кожен новий запуск гри може представляти щось нове для гравців.

Ці інструменти мають великий потенціал для створення процедурно генерованих рівнів у вашій грі. Вони надають розробникам різні засоби та можливості для реалізації та експериментування з процедурною генерацією рівнів у власних проектах.

2.2.2 Бібліотеки та Інструменти Розробки

Цей розділ присвячений аналізу та огляду найсучасніших бібліотек, які використовуються для створення та реалізації алгоритмів, спрямованих на процедурну генерацію віртуальних об'єктів та середовищ. В рамках цього дослідження буде вивчено функціональні можливості цих інструментів, їхню ефективність, а також способи використання та інтеграції з різноманітними програмними середовищами для розробки алгоритмів процедурної генерації. Цей розділ має на меті представлення широкого обсягу інформації про наявні інструменти та бібліотеки, які використовуються у сучасній практиці для створення алгоритмів процедурної генерації. Аналізуючи їхні можливості та особливості, ми зможемо зробити висновки щодо найбільш перспективних та ефективних інструментів для розробки алгоритмів процедурної генерації у віртуальних середовищах.

ProcGen, або процедурна генерація, - це метод створення вмісту, який використовує алгоритми для автоматичного генерування об'єктів, світів, сценаріїв або інших елементів у програмах та іграх. Цей підхід дозволяє створювати велику кількість різноманітного вмісту шляхом використання випадкових або напів випадкових алгоритмів, що робить кожен згенерований об'єкт унікальним. ProcGen широко застосовується у галузі розваг, зокрема у відеоіграх. Він дозволяє створювати безліч унікальних рівнів, мап, персонажів та інших об'єктів, що робить геймплей більш різноманітним та цікавим для гравців. Також procGen може застосовуватись у сферах комп'ютерної графіки, музики, генерації текстів, анімації та інших областях, де потрібно створювати великий обсяг контенту. Цей підхід має численні переваги, такі як зменшення рутинної роботи при створенні вмісту, збільшення різноманітності та унікальності згенерованого контенту, а також економія часу та ресурсів розробника. Однак важливо правильно налаштувати алгоритми, щоб

забезпечити баланс між випадковістю та створенням змістовного та привабливого контенту для користувача.

Dungeon Architect - це програмний інструмент чи плагін, який використовується для автоматизованої генерації дизайну підземеллів (данжів) у відеоіграх чи інших віртуальних середовищах. Його використовують розробники для створення складних ігрових рівнів, які складаються з лабіринтів, кімнат, коридорів та інших елементів, характерних для підземних локацій. Dungeon Architect надає можливість створювати цілком унікальні та різноманітні підземелля, використовуючи алгоритми процедурної генерації. Це дозволяє розробникам створювати велику кількість різноманітних ігрових рівнів без необхідності вручну моделювати або розміщувати кожен елемент вручну. Від Dungeon Architect можна очікувати різноманітні можливості, такі як налаштування параметрів генерації (таких як розмір кімнат, складність лабіринту тощо), контроль стилю й архітектури підземелля, а також можливість внесення змін або додавання різних додаткових елементів для призначення унікальності кожному створеному рівню. Цей інструмент допомагає розробникам прискорити процес розробки ігрових рівнів, забезпечуючи при цьому широкий спектр можливостей для створення цікавих та різноманітних підземних локацій у відеоіграх.

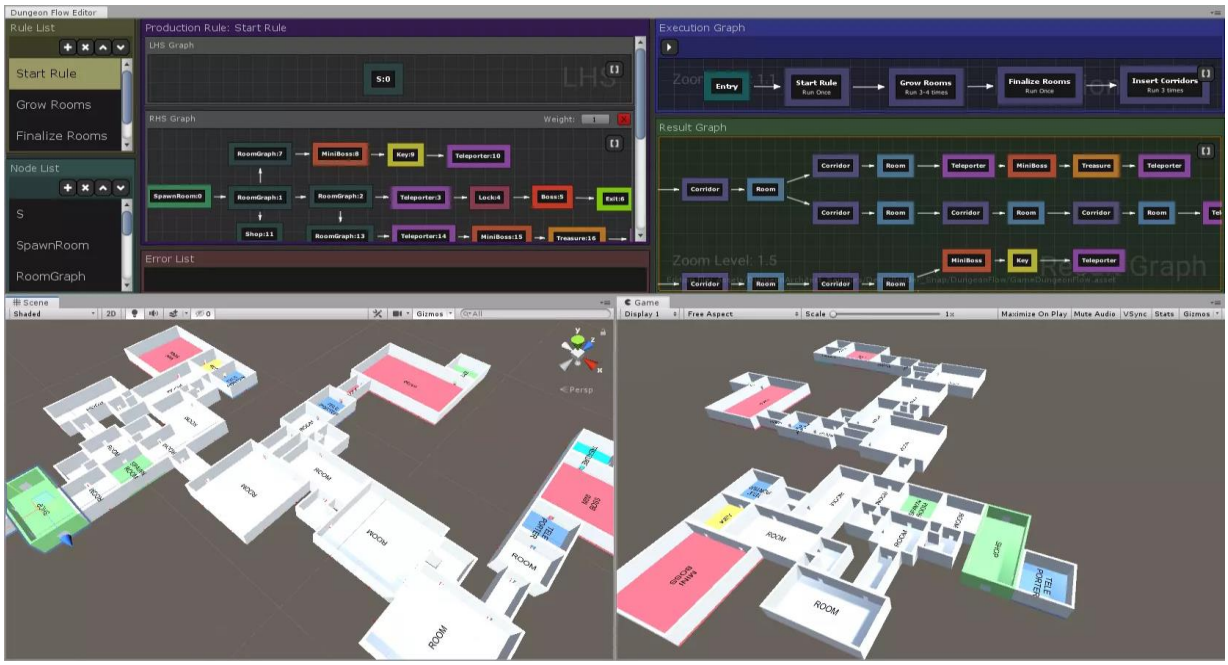

Рис 2.1. Приклад процедурної генерації в програмі Dungeon Architect

Houdini - це потужний інструмент для створення візуальних ефектів, анімації, 3D-моделювання, генерації контенту та процедурної генерації. Розроблений компанією SideFX, Houdini здатний створювати складні 3D-сцени, анімацію та спецефекти, що знаходять застосування у відеоіграх, фільмах, рекламі та інших сферах. Одна з головних особливостей Houdini - його процедурна природа. Це означає, що він базується на використанні вузлів (nodes) та графів, що дозволяє створювати складні системи зміни контенту згідно з параметрами та правилами, а не просто створювати моделі чи ефекти вручну.

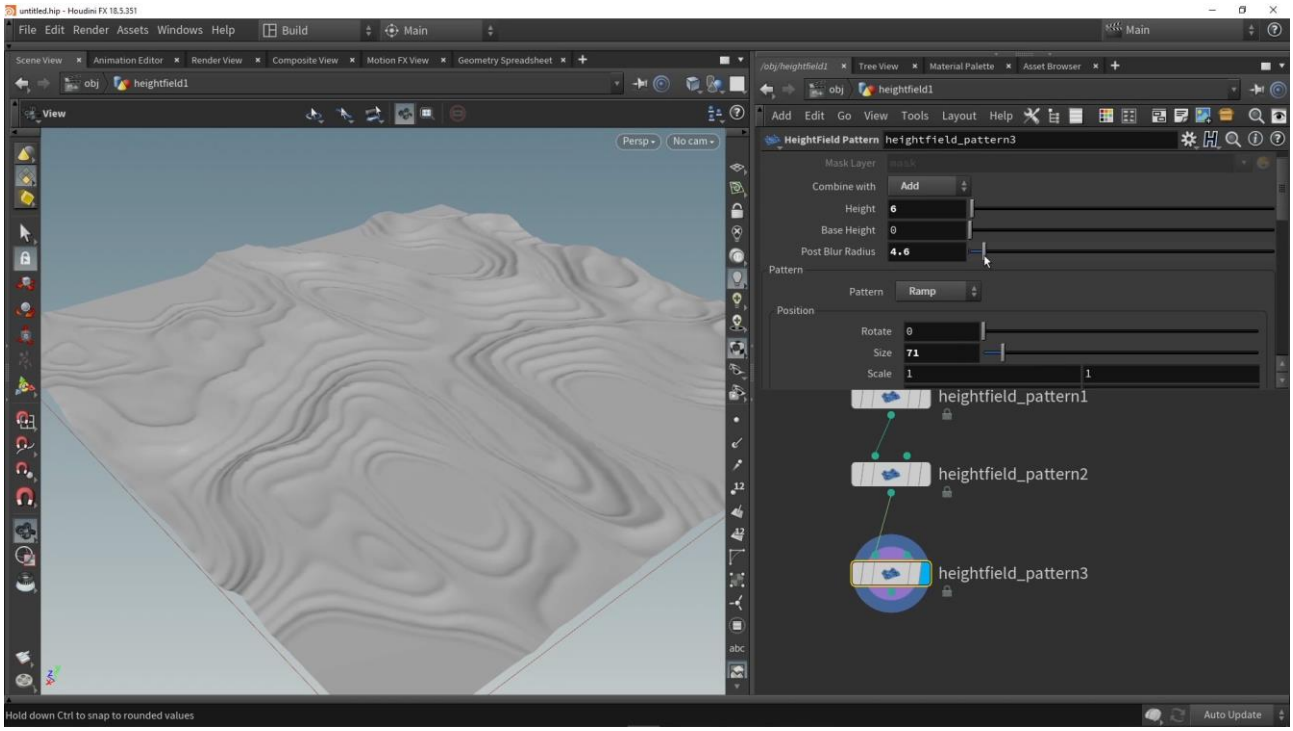

Рис 2.2. Процедурна генерація ландшафту в Houdini

Це надає безмежні можливості для генерації унікального, різноманітного й керованого контенту. У сфері геймдеву, Houdini використовується для створення різних аспектів гри, від генерації терейнів та рівнів до анімації персонажів та спецефектів. Він дозволяє розробникам швидко створювати та експериментувати з різноманітними концепціями, що полегшує процес розробки й сприяє інноваціям у створенні ігор. Houdini широко використовується у відомих голлівудських студіях для створення візуальних ефектів у фільмах, оскільки його гнучкість та потужність дозволяють реалізовувати навіть найскладніші концепції та задуми. Його графічний інтерфейс та широкі можливості роблять Houdini потужним інструментом для створення складних та креативних візуальних сцен, ефектів та геймплею.

Ці інструменти та бібліотеки надають розробникам широкий спектр можливостей для створення процедурно генерованих ігрових рівнів та елементів. Вони дозволяють реалізувати різноманітність і унікальність в ігровому дизайні та допомагають здійснити творчі ідеї у реальність.

2.2.3 Спеціалізовані Платформи та Інструменти

Алгоритм Wave Function Collapse (WFC) - це потужний метод процедурної генерації, який знаходить застосування у створенні унікальних шаблонів або патернів, що відповідають заданим обмеженням. Його основна ідея полягає в тому, щоб заповнити область даних відповідно до заданих правил, що забезпечує консистентність та логічність результатів. У методі WFC область даних, яку потрібно заповнити (наприклад, патерн для текстури чи рівня гри), розбивається на блоки або сітку. Кожен блок має можливість приймати різні значення відповідно до заданих обмежень та контексту сусідніх блоків. Алгоритм працює шляхом розглядання можливих комбінацій значень для кожного блоку, враховуючи обмеження та контекст. Він послідовно вибирає та обчислює значення для блоків, збігаючись з контекстом сусідніх блоків, забезпечуючи таким чином логічні та консистентні результати. Якщо певний блок не може бути узгоджено з контекстом, він може бути виключений або перевірений пізніше. WFC знайшов своє застосування у генерації різноманітних шаблонів для ігрових об'єктів, текстур, ландшафтів та інших елементів гри. Він дозволяє створювати складні та візуально захоплюючі мозаїки або структури, які відповідають заданим параметрам та обмеженням, використовуючи лише набір початкових правил та вихідних даних.

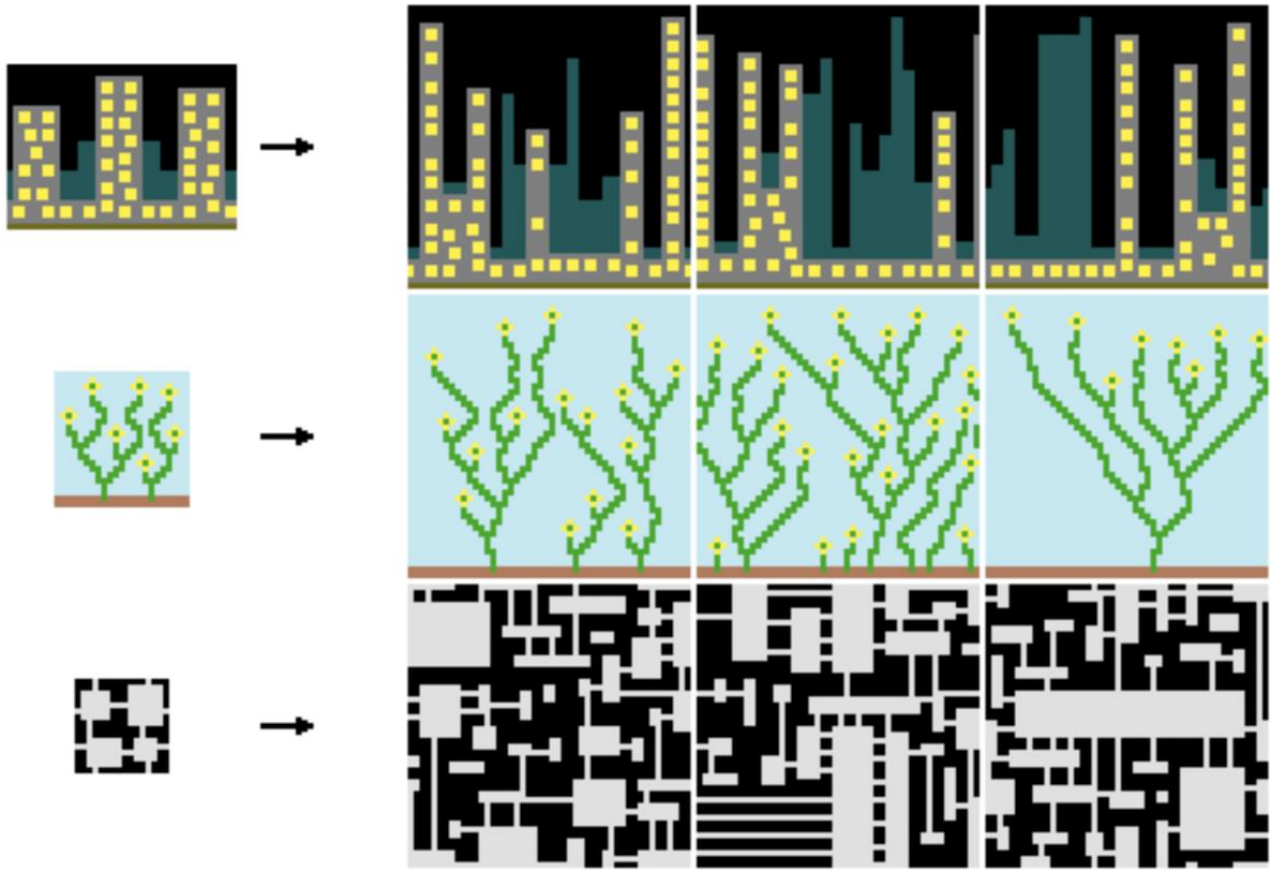

Рис 2.3. Приклад алгоритма Wave Function Collapse

Також є Unity's Procedural Generation Package - це арсенал інструментів, який Unity надає для підтримки процедурної генерації контенту в іграх. Він включає в себе кілька ключових складових, що спрямовані на підвищення продуктивності та швидкості розробки. C# Job System дає можливість оптимізувати швидкість обчислень за допомогою паралельних потоків. Burst Compiler допомагає перетворювати код, написаний на C#, в ефективний і виконуваний машинний код, що прискорює виконання програми. Entity Component System (ECS) створює ефективні системи обробки даних. Data-Oriented Technology Stack (DOTS) використовує ECS, C# Job System та Burst Compiler для оптимізації та прискорення розробки, зокрема, у процедурній генерації. Цей пакет інструментів Unity допомагає розробникам створювати швидкий та ефективний код для гри, зокрема в контексті процедурної генерації,

що сприяє покращенню продуктивності та розширює можливості створення унікального контенту для ігор.

2.2.4 Інтеграція та Продуктивність

API (Application Programming Interface) для процедурної генерації рівнів відіграє важливу роль у спільності розробників, дозволяючи їм створювати власні інструменти, програми та рішення, що використовуються для генерації геймплею. API для процедурної генерації може містити набір функцій, методів та протоколів, які розробники можуть використовувати для створення або інтеграції інструментів у свої проекти. Це дозволяє забезпечити спільний доступ до можливостей процедурної генерації та використання цих можливостей у різних інструментах та програмах. Інтеграція інструментів для процедурної генерації рівнів відбувається через використання цих API. Розробники можуть створювати власні редактори рівнів, інструменти для генерації контенту, плагіни для ігрових двигунів чи ігрових середовищ, використовуючи доступні API для процедурної генерації. Інтеграція забезпечує гнучкість у роботі з різними інструментами та програмами, сприяючи обміну даними, параметрами та рішеннями, що стосуються процедурної генерації. Це дозволяє розробникам створювати більш різноманітний та цікавий контент, використовуючи доступні ресурси та можливості. Інтеграція в контексті процедурної генерації рівнів означає об'єднання різних інструментів, платформ або систем для створення цілісного середовища для розробки віртуальних просторів. Це може охоплювати поєднання різних ігрових двигунів, інструментів для процедурної генерації, використання різних бібліотек чи API для створення складних інтерфейсів між різними програмними рішеннями. Мета інтеграції полягає в полегшенні розробки, підвищенні ефективності та розширенні можливостей розробників для створення різноманітного, цікавого та інноваційного контенту у віртуальних середовищах.

Розробка через процедурну генерацію рівнів може відкрити двері до безмежного світу творчості та можливостей для розробників відеоігор. Цей метод дозволяє автоматизувати процес створення великого обсягу ігрового контенту, від вулиць міст до підземних лабіринтів, використовуючи алгоритми та правила для генерації унікальних та різноманітних сценаріїв. Проте варто врахувати, що розробка та налаштування алгоритмів для процедурної генерації може вимагати значних зусиль і часу. Витрати ресурсів на створення цих алгоритмів та їхню оптимізацію можуть бути відчутними, особливо у випадку потреби у великій кількості деталізації або складних умовах генерації. Також важливо враховувати, що процедурна генерація може призвести до непередбачуваних результатів. Це може вимагати більш пристойного тестування та контролю якості, щоб упевнитися, що згенеровані рівні відповідають геймплею та не мають помилок, які могли б негативно вплинути на гру чи взаємодію гравців з нею.

Інтеграція процедурної генерації може бути важливим кроком у розробці відеоігор, але варто розглянути всі плюси та мінуси цього підходу, враховуючи ресурси, необхідні для успішного впровадження.

Оптимізація в контексті процедурної генерації рівнів є критичним етапом для досягнення бажаних результатів у відеоігровій розробці. Це процес оптимізації алгоритмів, структур даних та ресурсів, щоб забезпечити ефективну роботу програми та максимально використовувати потужності комп'ютера. Процедурна генерація може бути вимогливою до ресурсів, тому важливо розуміти ключові аспекти оптимізації для досягнення бажаних результатів у розробці ігор. Одним з основних аспектів оптимізації є ефективне управління ресурсами. Це означає ефективне використання пам'яті та обчислювальної потужності пристрою для мінімізації затрат та забезпечення плавної та безперебійної гри. Для досягнення бажаних результатів у процедурній генерації рівнів, важливо використовувати оптимальні алгоритми та структури даних.

Велика кількість обчислень чи складні алгоритми можуть призвести до зайвих навантажень на систему, тому вибір ефективних та оптимізованих методів генерації важливий для досягнення успіху. Ще одним ключовим фактором є попереднє та постійне тестування. Тестування дозволяє виявити можливі помилки в роботі алгоритмів, низьку продуктивність та можливість покращення. Після виявлення проблемних моментів можна внести відповідні зміни, щоб оптимізувати процедурну генерацію та підвищити ефективність її роботи. Уникнення зайвих обчислень та використання мінімальної кількості ресурсів, вибір оптимальних алгоритмів та структур даних, а також постійне тестування для виявлення та усунення проблем - це ключові компоненти оптимізації процедурної генерації рівнів. Відправляючись цим шляхом, розробники можуть досягти бажаних результатів у створенні ігор, що вражають своєю унікальністю, геймплеєм та оптимізацією.

Реалізація потреб конкретного проекту у відеоігровій розробці вимагає комплексного підходу та уважного розгляду деталей для досягнення поставлених цілей. Особливості реалізації визначаються специфікою проекту, його концепцією, цілями та вимогами. Розглянемо, наприклад, створення великої відкритої гри з високим рівнем іммерсії та глибоким геймплеєм.

Уявімо, що це проект нової відеоігри, яка висувається як відкритий світ з великою кількістю локацій, складних квестів та великою варіативністю геймплею. Ця гра спрямована на глибоке поглиблення гравців у віртуальний світ, створення власних історій та взаємодію з навколишнім середовищем.

Реалізація цього проекту вимагатиме потужних технологій для створення відкритого світу з великою кількістю деталей. Використання високоефективних геймдвижків, наприклад, Unreal Engine чи Unity, може забезпечити потрібні можливості для створення відмінної графіки та реалістичної фізики.

Для досягнення великої кількості різноманітних локацій та квестів у великому відкритому світі, процедурна генерація може стати ключовим

інструментом. Вона дозволить автоматизувати створення окремих елементів середовища та завдань, забезпечуючи грати нескінченну кількість можливостей для гравців.

Реалізація проекту потребує великої кількості різноманітного контенту - від моделей персонажів та об'єктів до текстур, діалогів та музики. Важливо створити динамічний та змістовний геймплей, враховуючи інтерактивність імовірних сценаріїв. Тестування та Оптимізація Завершальний етап розробки передбачає великий обсяг тестування для виявлення помилок, вдосконалення геймплею та оптимізації продуктивності гри на різних пристроях.

Реалізація конкретного проекту у відеоігровій індустрії - це складний, але захоплюючий процес, що потребує багато уваги до деталей, креативності та технічної експертизи. Успішна реалізація потреб проекту забезпечить глибокий імерсивний досвід для гравців та створить вражаючий світ для їхніх пригод.

2.3 Аналіз Сучасних Практик Процедурної Генерації Рівнів

Процедурна генерація рівнів у відеоіграх - це захоплюючий світ інновацій, який надає нескінченні можливості для створення унікальних ігрових угор, що залишаються непередбачуваними та захоплюючими для гравців. Цей метод спирається на використання алгоритмів та правил для автоматизації процесу створення ігрового контенту, що дозволяє уникнути монотонності та рутинності ручного моделювання і відкриває двері до нового індивідуального ігрового досвіду кожного разу, коли гравець починає гру. Заснована на алгоритмах, процедурна генерація надає можливість швидко створювати унікальні об'єкти, рівні та ландшафти, роблячи гру більш динамічною і непередбачуваною. Основною перевагою цього підходу є здатність відтворювати нескінченну кількість варіантів ігрового середовища, що стимулює інтерес гравців та збільшує переігровуваність гри. Проте, існують виклики, пов'язані з

ефективністю та якістю генерації. Необхідно уникати створення занадто простих чи складних рівнів, забезпечуючи баланс між викликом та задоволенням. Більше того, недосконаліть алгоритмів може призвести до монотонності відтворюваних сценаріїв чи обмеженої різноманітності. Процедурна генерація рівнів - це багатогранна техніка, яка потребує балансування між автоматизацією та творчістю, забезпечуючи гармонію між унікальністю та якістю, щоб забезпечити унікальний та цікавий геймплей для гравців.

Сучасні ігри активно використовують процедурну генерацію для створення унікальних та різноманітних ігрових середовищ. Це дозволяє кожному гравцеві отримати унікальний досвід гри, оскільки кожен рівень генерується автоматично.

Процедурна генерація рівнів забезпечує динамічний геймплей, де події та ситуації можуть змінюватися при кожному проходженні. Це збільшує реіграбельність та інтерес до гри, оскільки гравці ніколи не знають, що очікувати.

роцедурна генерація дозволяє ефективно використовувати ресурси, оскільки вона може створювати великі об'єми контенту без потреби в ручному створенні кожного елементу. Це зменшує час розробки та витрати на створення ігрових рівнів. Адаптивність до Гравців Сучасні практики у процедурній генерації спрямовані на створення геймплею, який адаптується до стилю гри кожного гравця. Це означає, що гра може адаптуватися до вмінь та вподобань гравців, що робить досвід більш персоналізованим.

Незважаючи на багато переваг, процедурна генерація рівнів має свої виклики, такі як потреба у балансуванні складності, уникненні монотонності та підтримці якості гри. Майбутні напрямки розвитку включають розробку більш складних алгоритмів, що створять ще більш реалістичні та цікаві ігрові світи. Аналіз сучасних практик процедурної генерації рівнів свідчить про те, що ця

технологія продовжує еволюціонувати, надаючи ігровій індустрії нові можливості та перетворюючи спосіб, яким ми розуміємо створення ігор.

РОЗДІЛ 3
РОЗРОБКА ТА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ЗАСТОСУВАННЯ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ РІВНІВ В ІГРОВИХ ЗАСТОСУНКАХ

3.1 Технології розробки програмного забезпечення

Для розробки та аналізу ефективності було прийнято рішення використовувати ігровий двигун Unity. Unity надає інструменти для тестування та аналізу рівнів, що створені процедурно. Це дозволяє швидко оцінювати їх ефективність та вносити зміни для поліпшення геймплею. Відповідно мова програмування була вибрана C#. Unity використовує мову програмування C# через низку переваг, які вона надає для розробки ігор та інтерактивних додатків.

Створення мови програмування C# (рис. 3.2)розпочалося у грудні 1998 року та готувалося для випуску разом із продуктами групи Millenium. Проект був позначений як COOL (C-style Object Oriented Language) і розроблявся як альтернатива Java від компанії Oracle. C# був оголошений основною мовою платформи Microsoft .Net Framework у 2000 році. У тому ж році з'явилася перша загальнодоступна бета-версія. Перша остаточна версія мови програмування C# була випущена у 2002 році разом з інтегрованою середою розробки програмного забезпечення Visual Studio .Net. Аналогічно до Java, C# втілив наступні концепції: Віртуальна машина: платформа .Net виконує програму подібно віртуальній машині Java. Байт-код: програмний код компілюється в проміжний мовний код MSIL (Microsoft Intermediate Language), а потім перетворюється на машинний код в залежності від платформи, на якій запускається програма. Керований код: оскільки програми, написані на C#, запускаються виключно в віртуальному середовищі CLR (Common Language Runtime), можна контролювати їх виконання і зупиняти в будь-який момент, а також керувати

використанням пам'яті, за необхідності виділяючи або звільняючи частини пам'яті, які використовує програма.

Рис 3.1 Логотип C#

По-перше, C# є високорівневою мовою, що дозволяє розробникам ефективно створювати складні програми завдяки широкому спектру функцій та інструментів.

По-друге, ця мова має розширену функціональність, вона дозволяє доступ до різноманітних бібліотек та фреймворків, що полегшує створення ігор з різноманітними можливостями.

По-третє, C# має інтеграцію з Unity, тобто API Unity працює добре з цією мовою, що спрощує взаємодію з різними компонентами двигуна гри. Додатково, популярність C# серед розробників забезпечує наявність широкої спільноти, де можна знайти підтримку, документацію та взаємодію з іншими спеціалістами. Крім того, кросплатформеність C# дозволяє розробникам створювати програми,

які працюватимуть на різних пристроях без необхідності написання окремого коду для кожної платформи.

У випадку Unity(рис. 3.2), мова програмування C# є основною мовою для розробки ігор і додатків. Unity створений з урахуванням того, що C# буде основною мовою програмування. Це означає, що багато функцій, можливостей та API Unity спроектовані для використання саме з цією мовою. Unity оптимізований для роботи з C#, що впливає на продуктивність розробки. Інтеграція між платформою і мовою програмування сприяє швидкому розробленню, тестуванню та впровадженню рішень.

Рис 3.2 Логотип Unity

За основу для кваліфікаційної роботи я вирішив розробити систему як буде генерувати рівні (тунелі) для гри типу topdown шутера.

Це жанр ігор, де гравець спостерігає за дією зверху вниз, отримуючи пташиний погляд на ігровий світ. Це подібно до погляду з висоти або погляду зверху на своїх персонажів та оточуючі області. Основна ідея полягає у керуванні персонажем або об'єктом, який рухається по рівню, стріляє ворогів або виконує різні завдання. У цьому жанрі можуть бути відмінності: кут огляду зверху вниз або збоку, але загальна ідея залишається тією ж. В основі лежить взаємодія гравця з ворожими силами, використання зброї та навичок для перемоги та рух по рівню. Гравець керує персонажем, який має можливість рухатись у будь-

якому напрямку, уникати перешкод, взаємодіяти з оточуючим середовищем та знаходити приховані об'єкти. Цей жанр вимагає поєднання стратегічних рішень та швидкого реагування. Гравцеві необхідно постійно приймати рішення, борючись з ворожими силами, які можуть бути численними та атакувати одночасно. Тут важливо поєднувати стратегічне планування з миттєвими реакціями на ситуацію. Деякі гри цього жанру також дозволяють гравцям об'єднуватися в команди для спільного вирішення завдань чи подолання викликів. Вони можуть бути як класичними аркадними іграми, де ви боретесь з ворогами на кожному рівні, так і більш сучасними іграми зі складним сюжетом та глибокими механіками гри. Топ-даун шутери стали популярними завдяки своїй простоті та в той же час складній грі, яка вимагає від гравця не тільки швидкості реакції, але й стратегічного мислення.

Рис 3.3 Приклад гри “Топ-даун”

На початку роботи було створено деякі об’єкти для того щоб система могла взаємодіяти між ними та використовувати ці об’єкти для подальшого будування та генерації того чи іншого рівня різної складності.

3.2 Огляд моделей та їх будова

Программа створена таким чином, що вона використовує блоки, які заздалегідь були створені за допомогою Unity з базових форм. За колір для фігур відповідає заздалегідь створені матеріали. Матеріал (рис. 3.13) в Unity - це налаштування, що визначає зовнішній вигляд об'єкту. Він контролює текстури, кольори, відблиски, прозорість та інші властивості об'єкту або поверхні. Матеріал дозволяє задати, як об'єкт відображає світло і як він реагує на нього. У матеріалі можна налаштувати різноманітні параметри, такі як текстури (зображення, що використовуються для візуального оформлення), бамп-мапи (для створення візуального ефекту вибивання деталей на поверхні), налаштування освітлення, прозорість, відблиски та багато іншого. Ці параметри дають можливість створювати різноманітні та реалістичні візуальні ефекти для об'єктів у вашій грі або програмі. Також, матеріали можна комбінувати з шейдерами - програмами, які контролюють процес рендерингу об'єктів, щоб створювати складні ефекти та зовнішній вигляд об'єктів, використовуючи різні властивості світла, тіней та колірних ефектів.

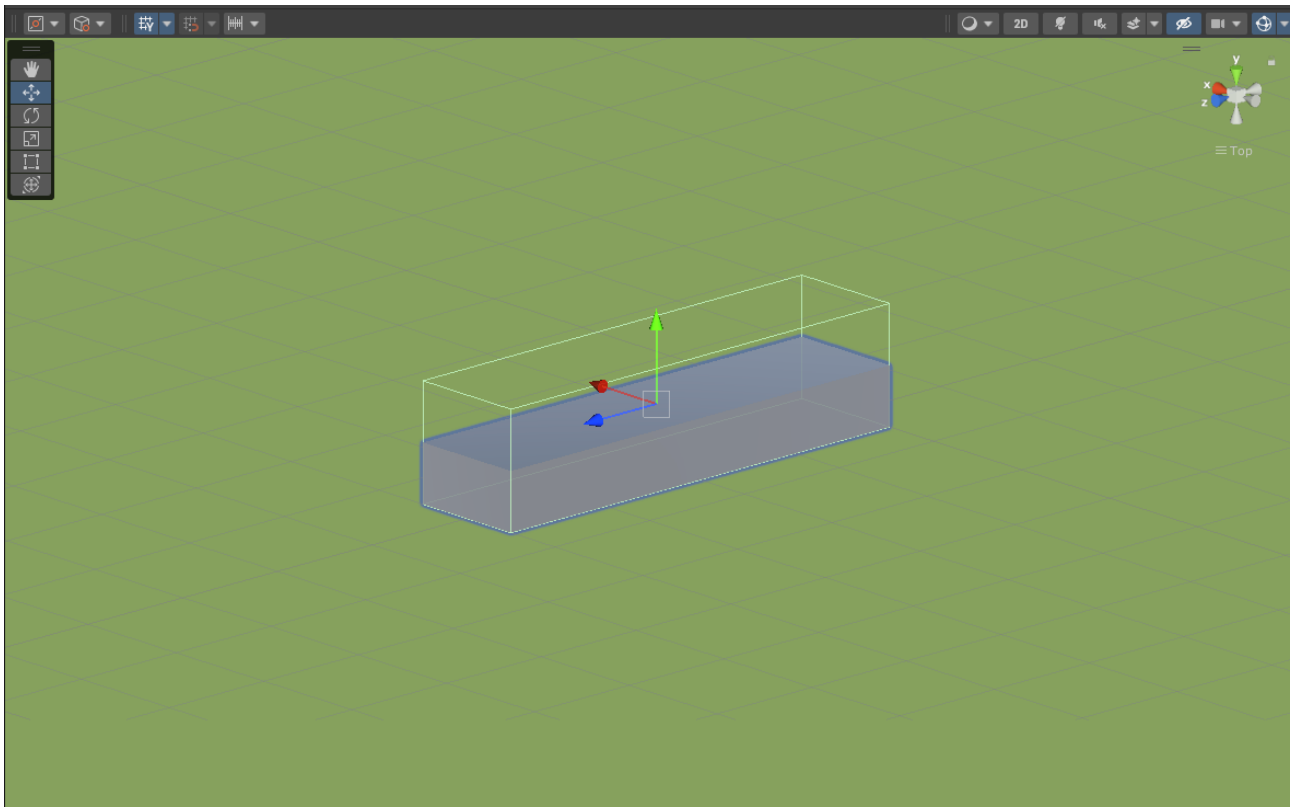

Рис 3.4 Блок “Коридор 1”

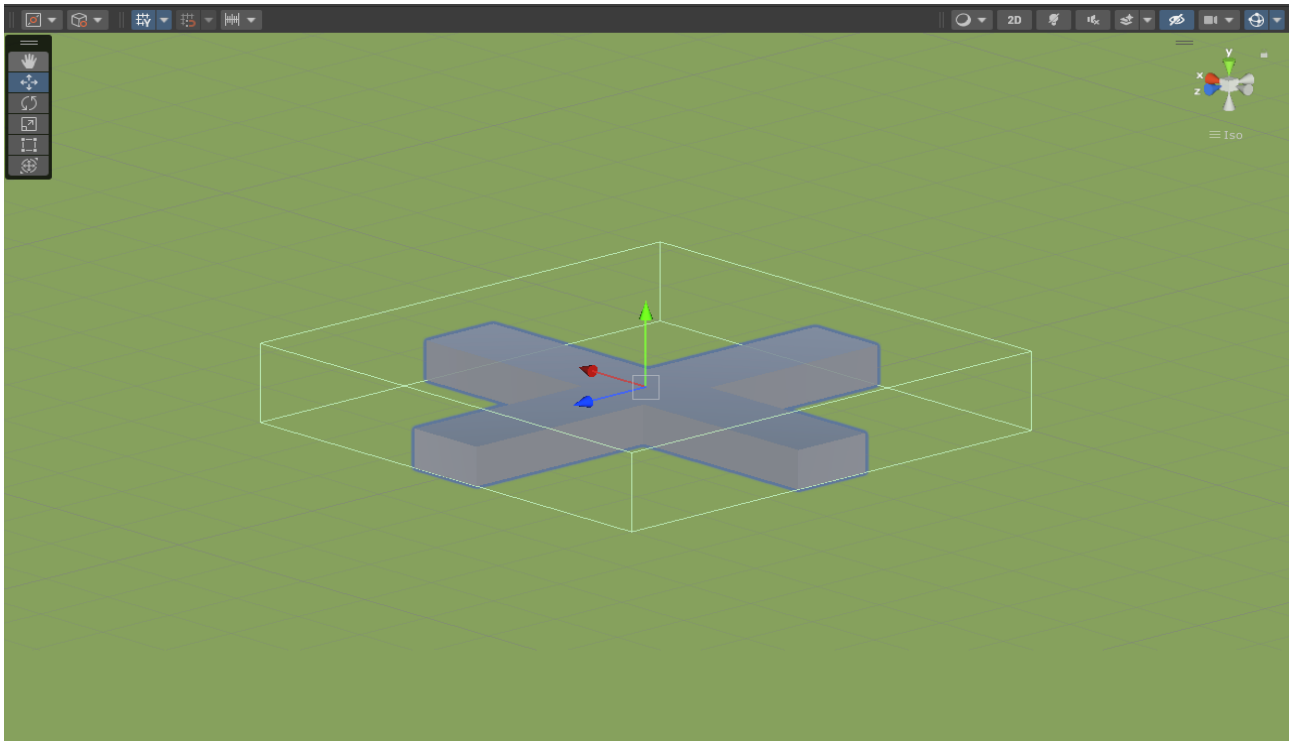

Рис 3.5 Блок “Коридор 2”

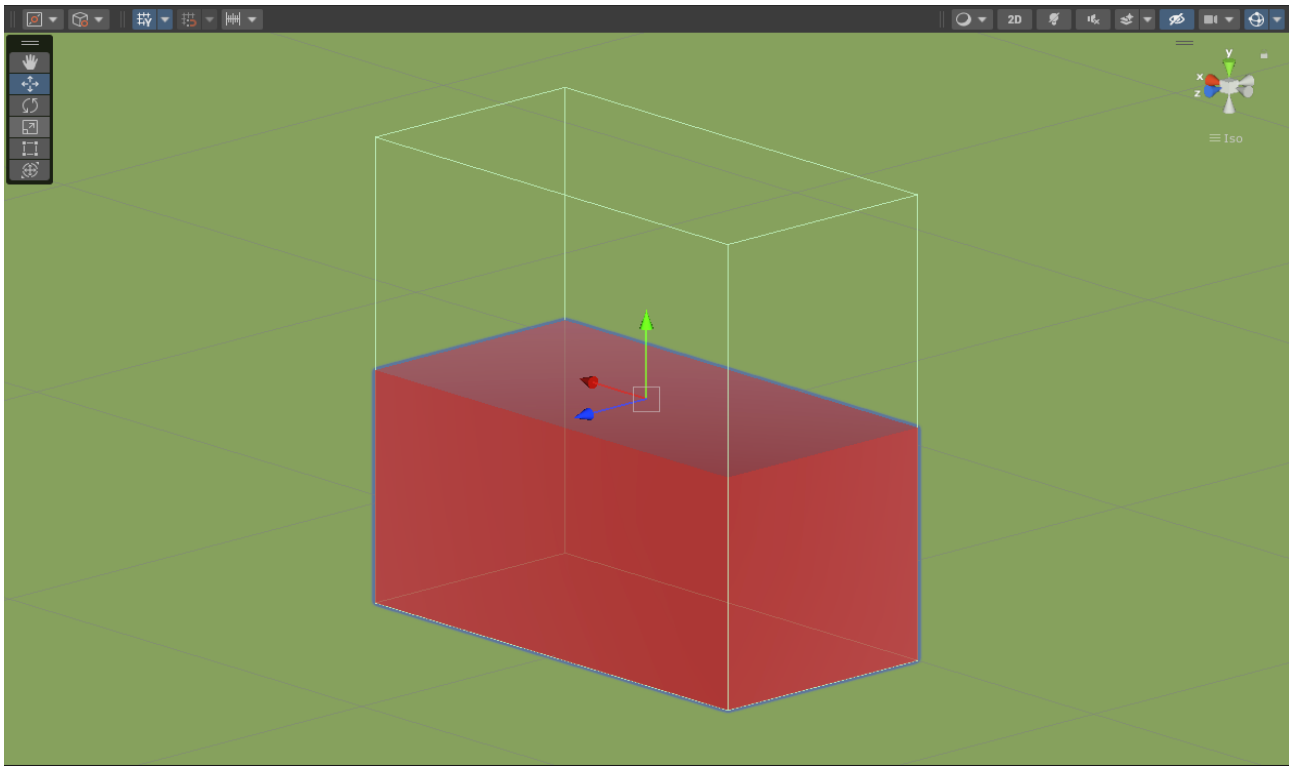

Рис 3.6 Блок “Пастка”

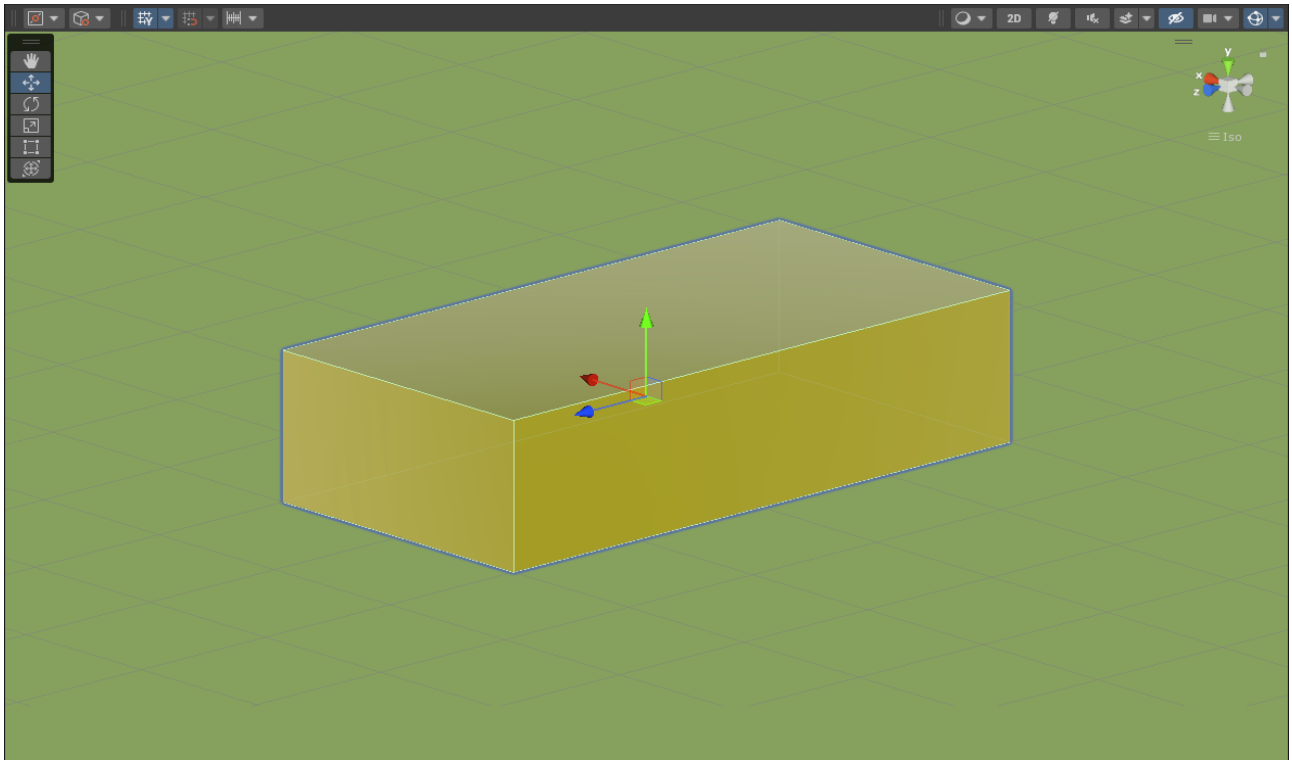

Рис 3.7 Блок “Коридор 1”

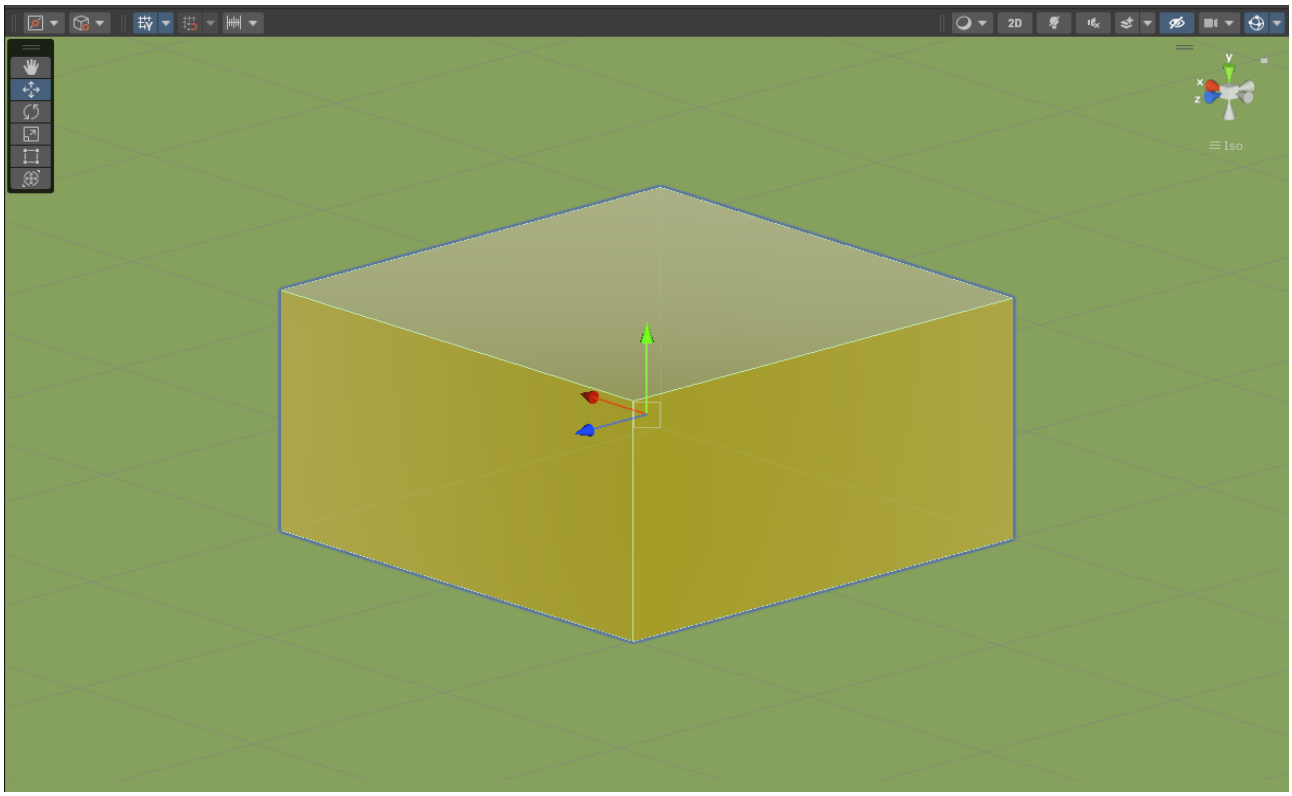

Рис 3.8 Блок “Коридор 1”

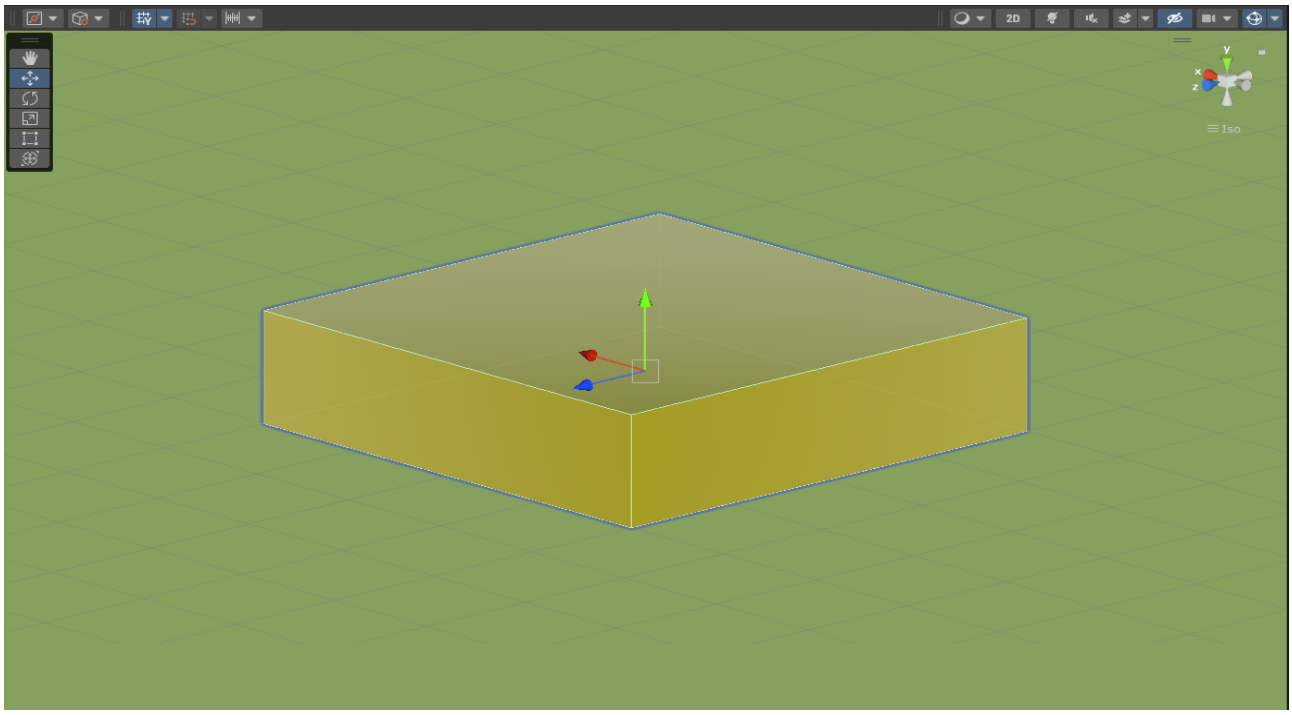

Рис 3.9 Блок “Коридор 1”

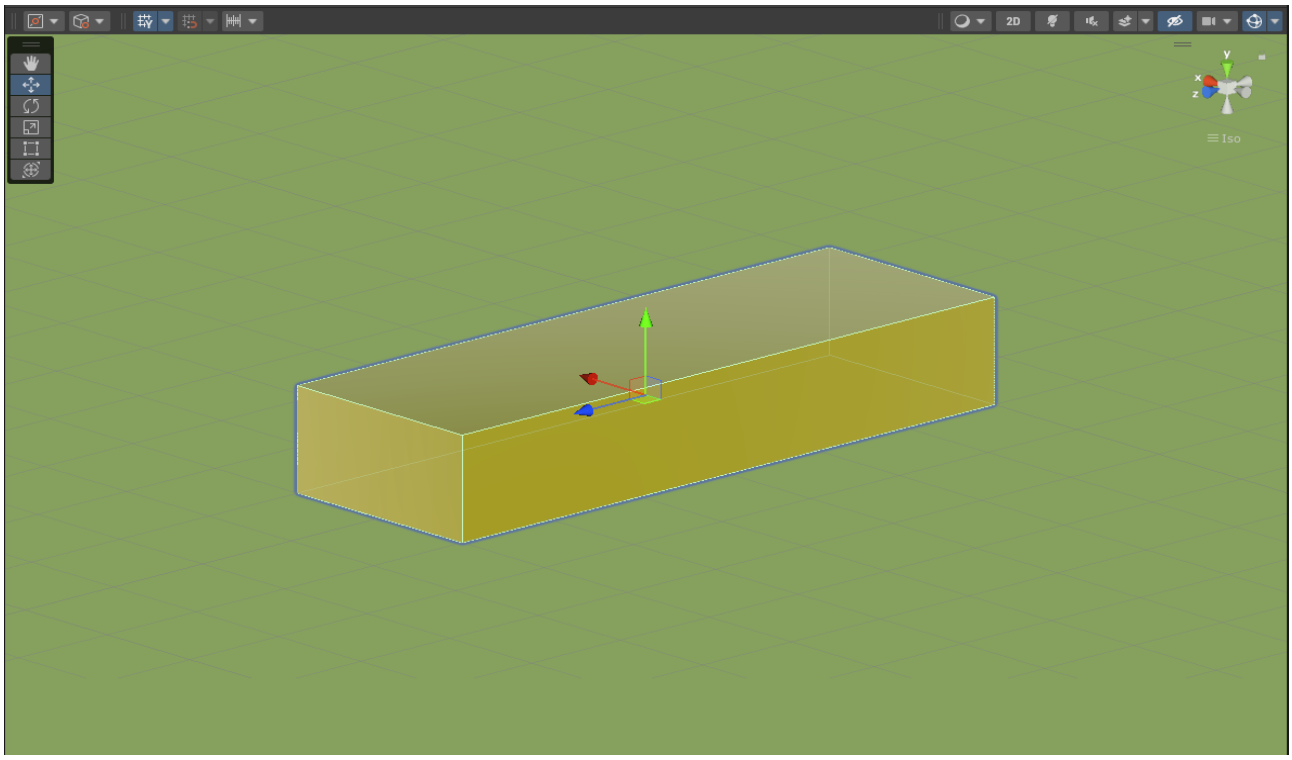

Рис 3.10 Блок “Коридор 1”

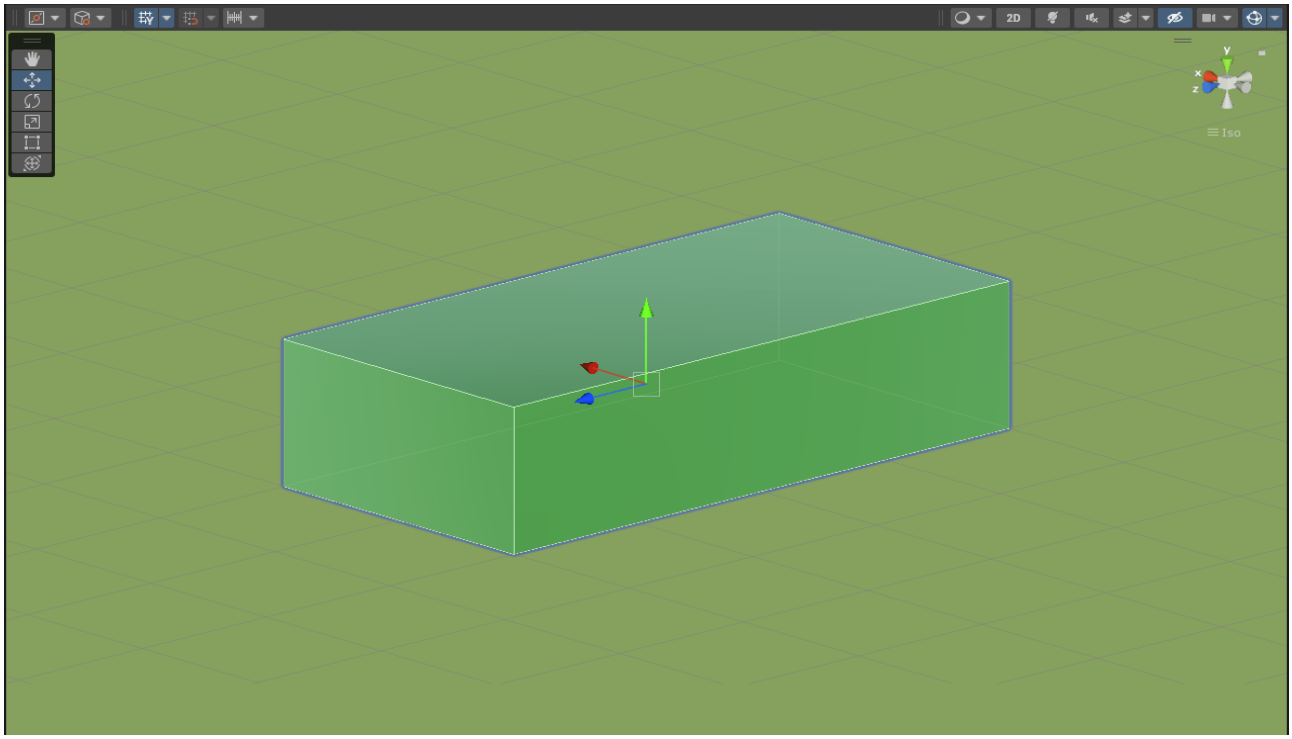

Рис 3.11 Блок “Коридор 1”

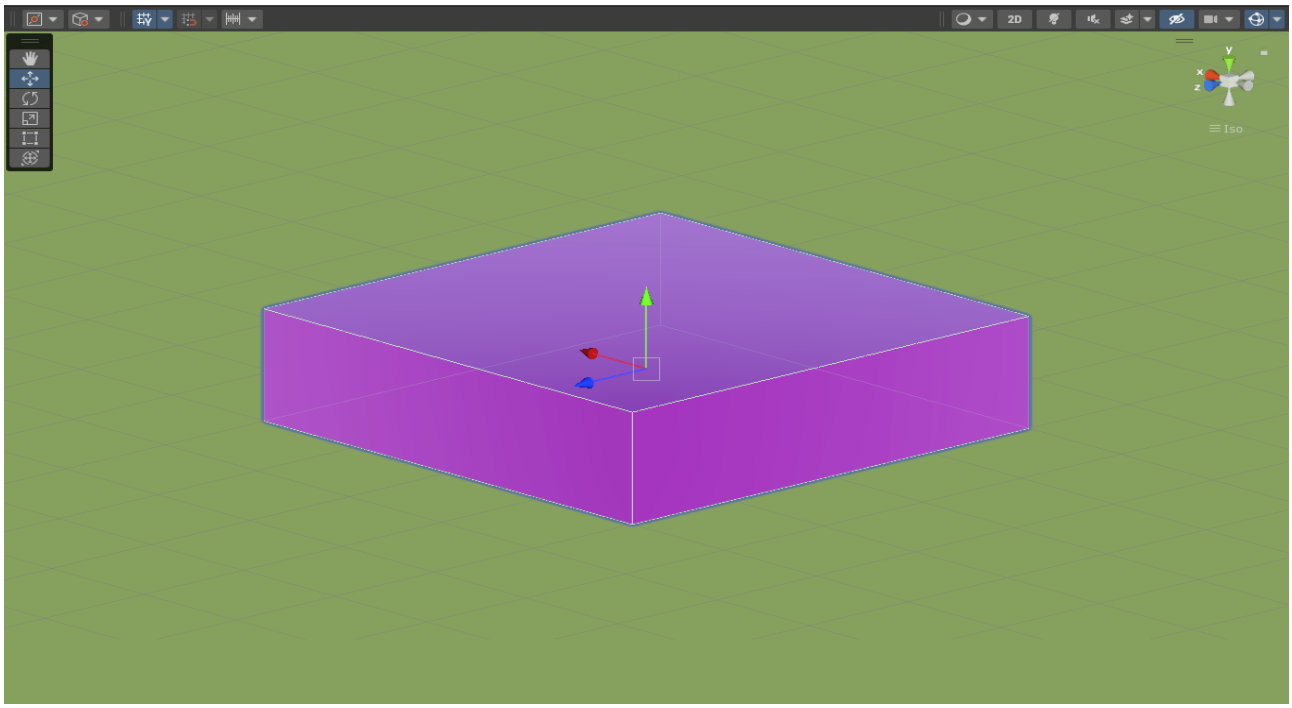

Рис 3.12 Блок “Коридор 1”

Программа створена таким чином, що вона використовує блоки, які заздалегідь були створені за допомогою Unity з базових форм. За колір для фігур відповідає заздалегідь створені матеріали. Матеріал (рис. 3.13) в Unity - це налаштування, що визначає зовнішній вигляд об'єкту. Він контролює текстури, кольори, відблиски, прозорість та інші властивості об'єкту або поверхні. Матеріал дозволяє задати, як об'єкт відображає світло і як він реагує на нього. У матеріалі можна налаштувати різноманітні параметри, такі як текстури (зображення, що використовуються для візуального оформлення), бамп-мапи (для створення візуального ефекту вибивання деталей на поверхні), налаштування освітлення, прозорість, відблиски та багато іншого. Ці параметри дають можливість створювати різноманітні та реалістичні візуальні ефекти для об'єктів у вашій грі або програмі. Також, матеріали можна комбінувати з шейдерами - програмами, які контролюють процес рендерингу об'єктів, щоб створювати складні ефекти та зовнішній вигляд об'єктів, використовуючи різні властивості світла, тіней та колірних ефектів.

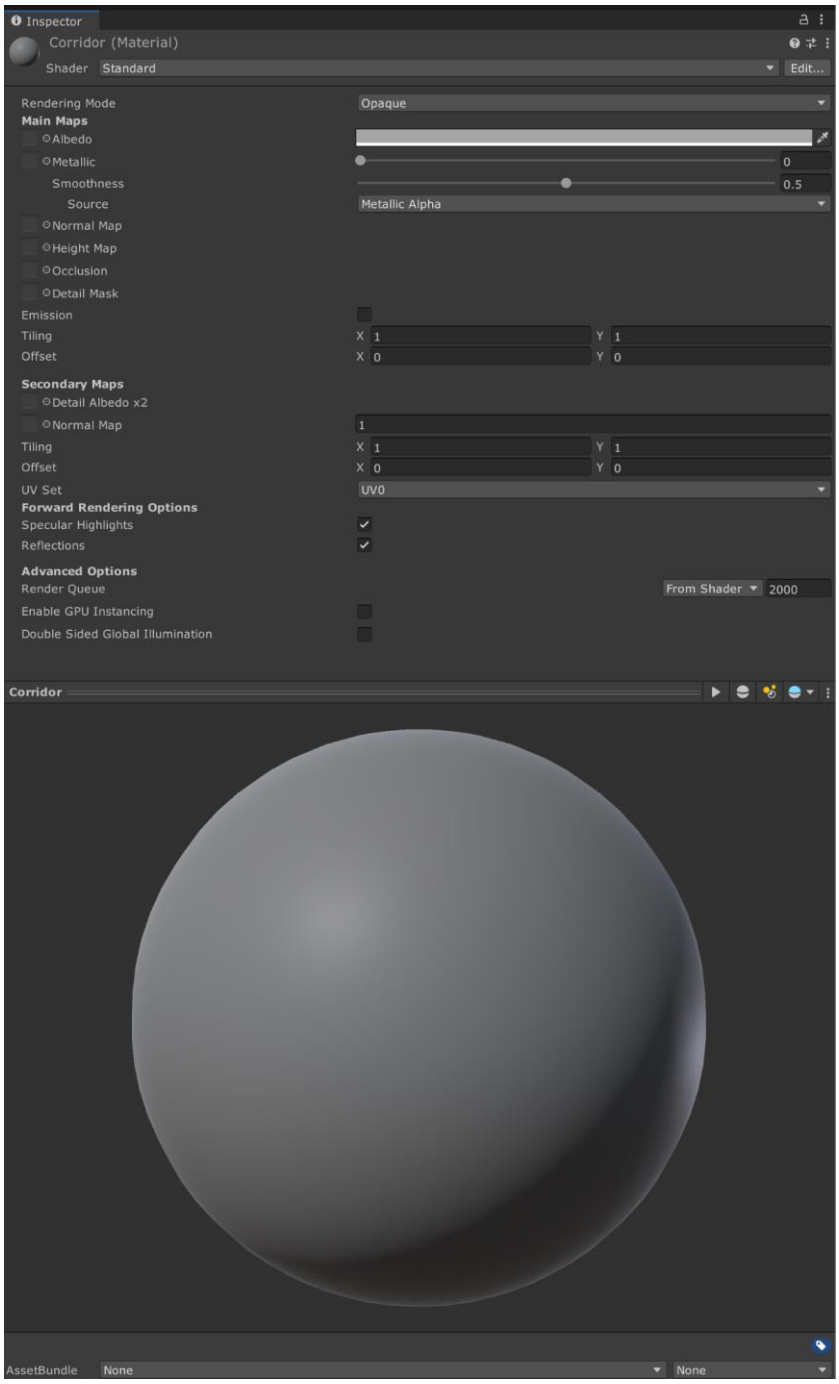

Рис 3.13 Матеріал та його параметри в Unity

Матеріал в Unity взаємодіє з моделлю через процес рендерингу. Кожна модель (або об'єкт) у віртуальному просторі може мати призначений матеріал, який визначає зовнішній вигляд цієї моделі. Коли Unity рендерить сцену, він застосовує матеріал до кожної поверхні моделі, щоб відобразити

відповідний вигляд. Це включає в себе властивості матеріалу, такі як текстури, колір, відблиски, прозорість та інші параметри, які впливають на те, як модель виглядає при візуалізації в грі або програмі.

3.3 Огляд роботи програми

При запуску програми, вона за допомогою кодової частини починає процес генерації рівня за тими параметрами які ми вказано в налаштуванні самого рівня(рис. 3.14).

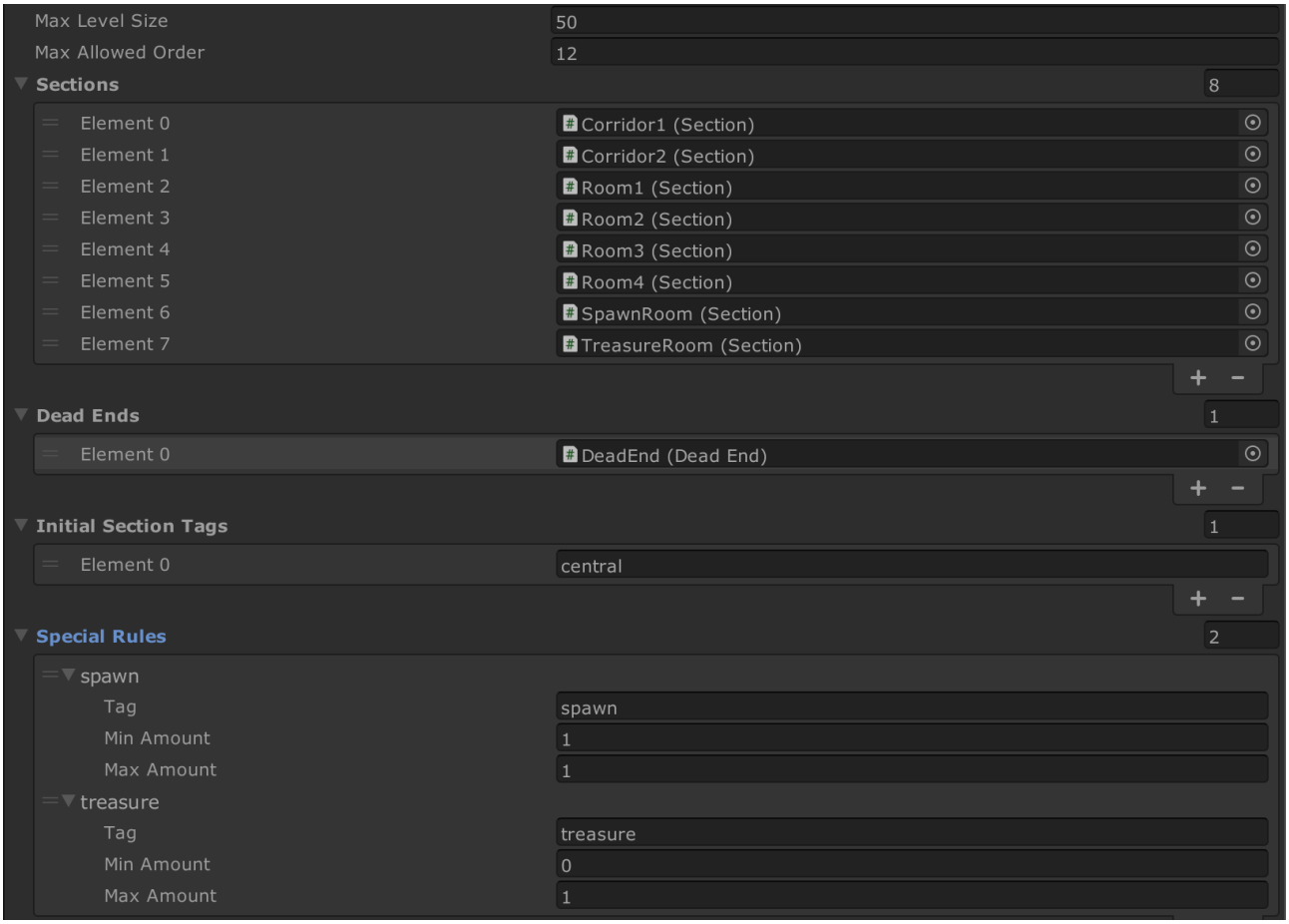

Рис. 3.14 Налаштування параметрів для генерації рівня

Також паралельно з цим запускається процес який буде рахувати який час пішов на виконання процедурної генерації рівня. Час виконання програми буде рахуватися в мілісекундах для подальшої точності.

Оглянемо параметри які надає програма і які є можливість налаштовувати для потреб генерації рівня.

1. Max Level Size - Максимальна кількість розділів для створення. Має числове значення(int).
2. Max Allowed Order : максимальна довжина (у секціях) гілок від оригінальний розділ. Встановлення великого числа призведе до довших шляхів, а встановлення коротшого призведе до структури, схожої на вулик
3. Sections (list of sections) : розділи будуть завантажені як префаби в цю колекцію
4. Dead Ends (list of dead ends) : збірні тупики будуть завантажені як збірні у цій колекції
5. Initial Section Tags (список рядків): теги, які визначатимуть, який розділ використовується як початкова частина рівня.
6. Special Rules (список правил): тут ми визначимо спеціальні правила, щоб примусово виконувати деякі аспекти процесу генерації.

Кожен запуск програми буде генеруватись новий рівень.

3.4 Результати процедурної генерації рівнів за різними параметрами

Варіант №1:

Параметри:

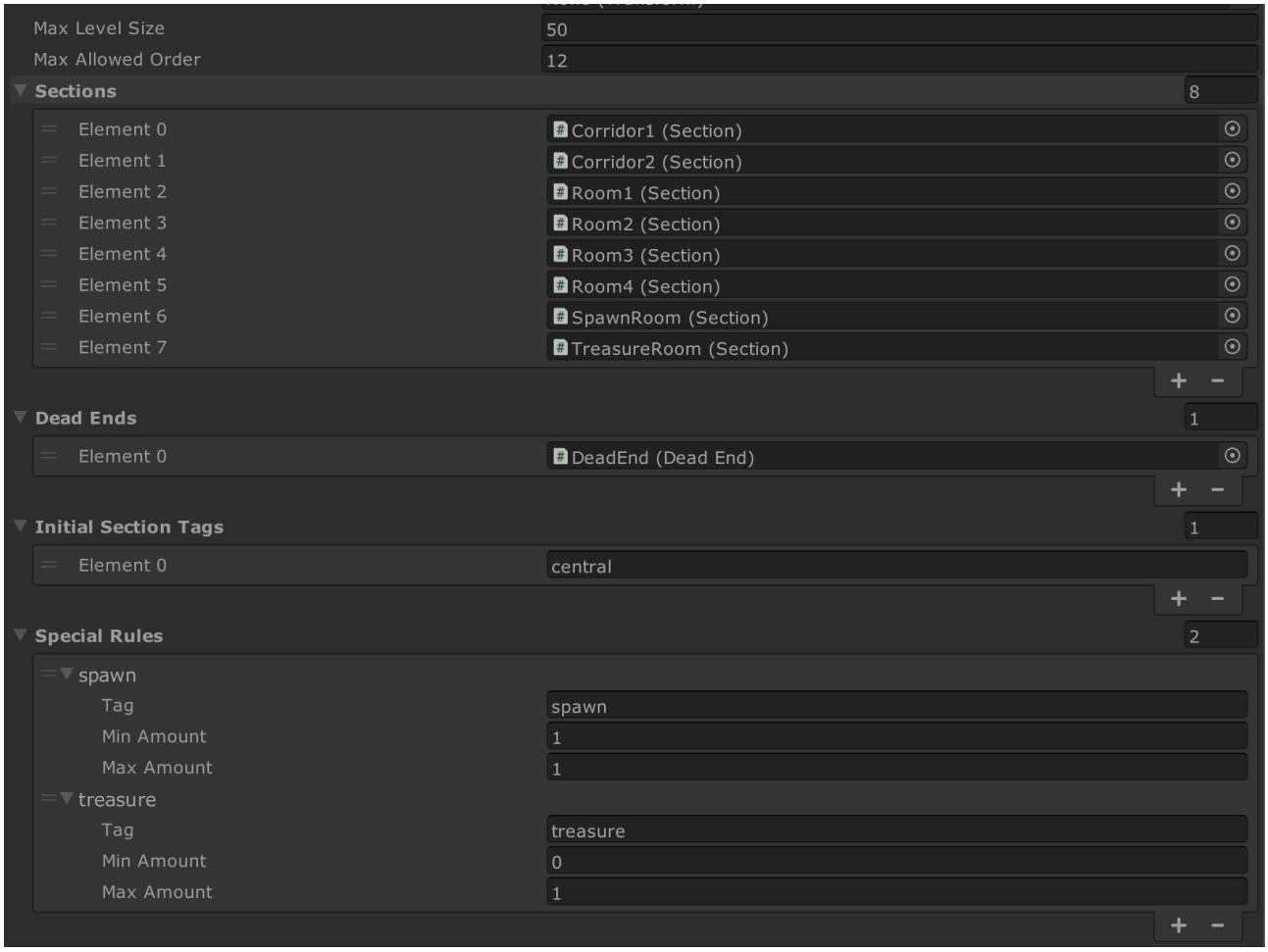

Рис 3.15 Параметри варіанта 1

Результат:

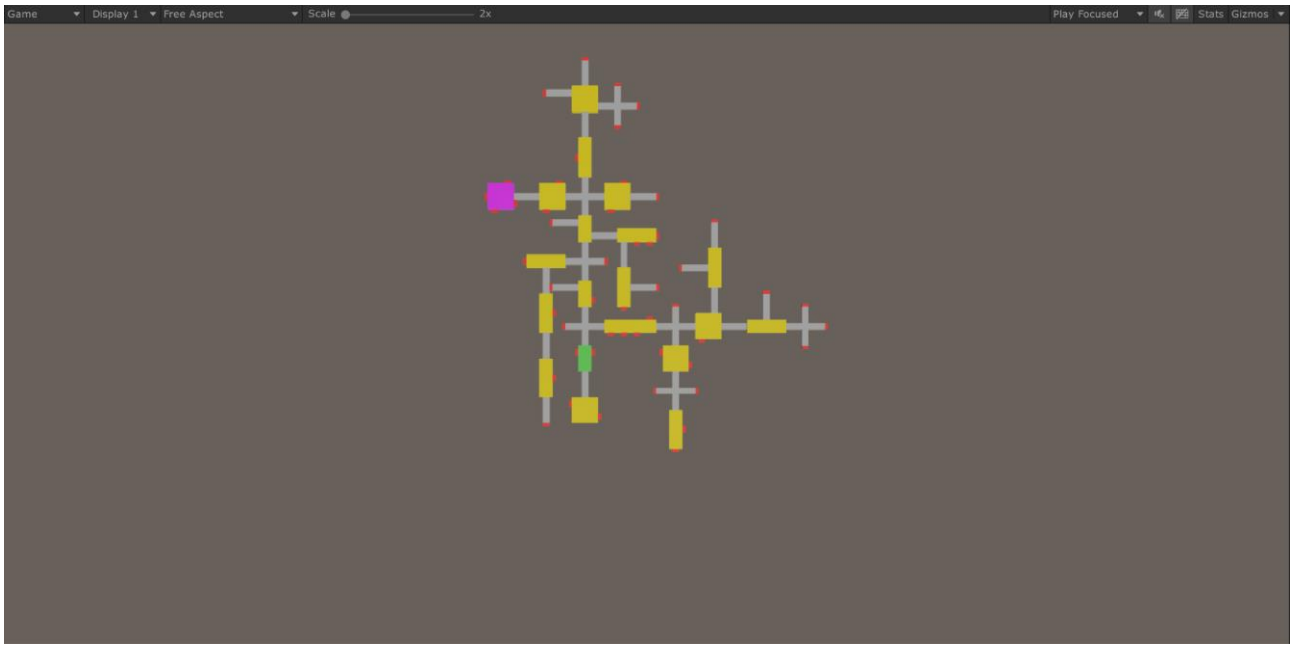

Рис 3.16 Результат варіанта 1

Варіант №2

Параметри:

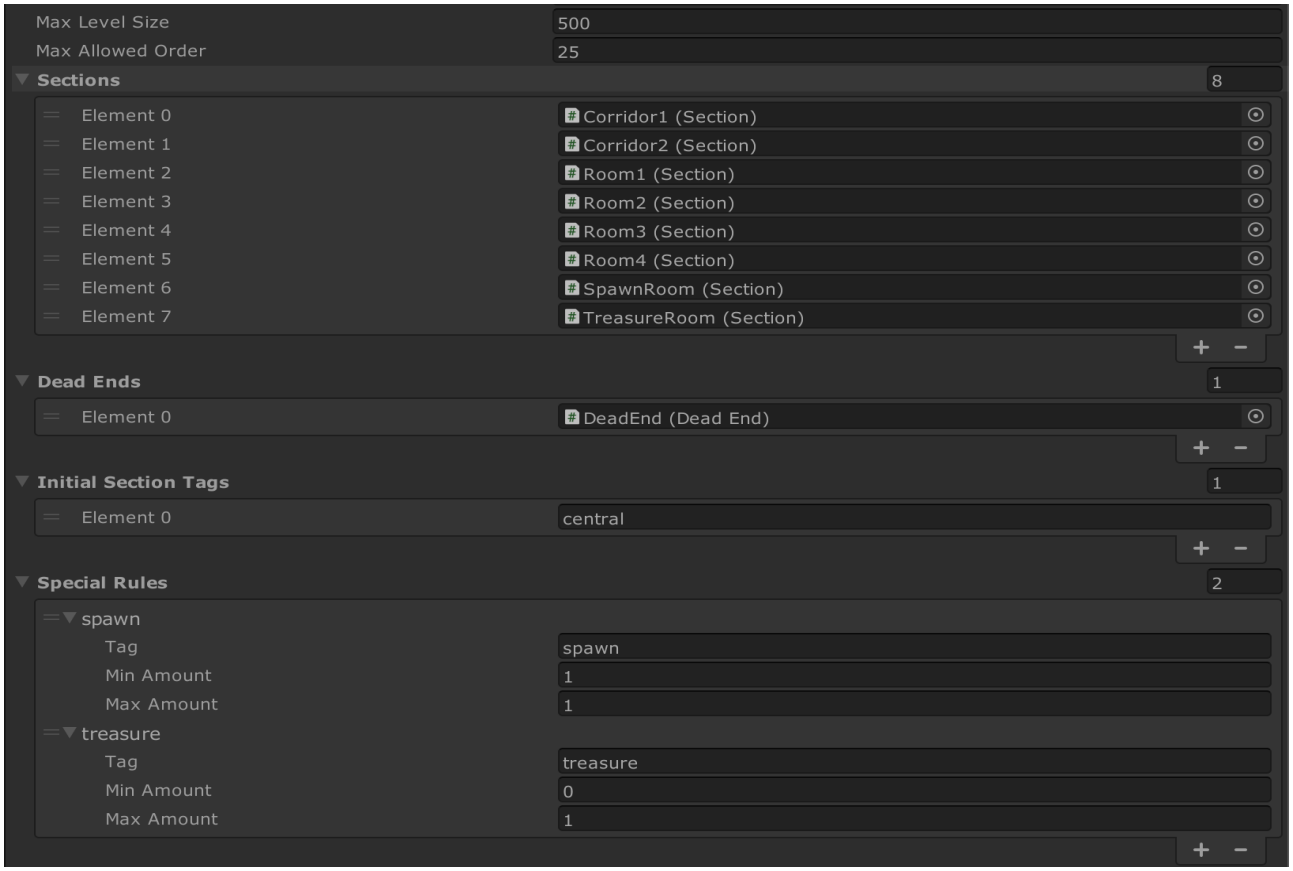

Рис 3.17 Параметри варіанта 2

Результат:

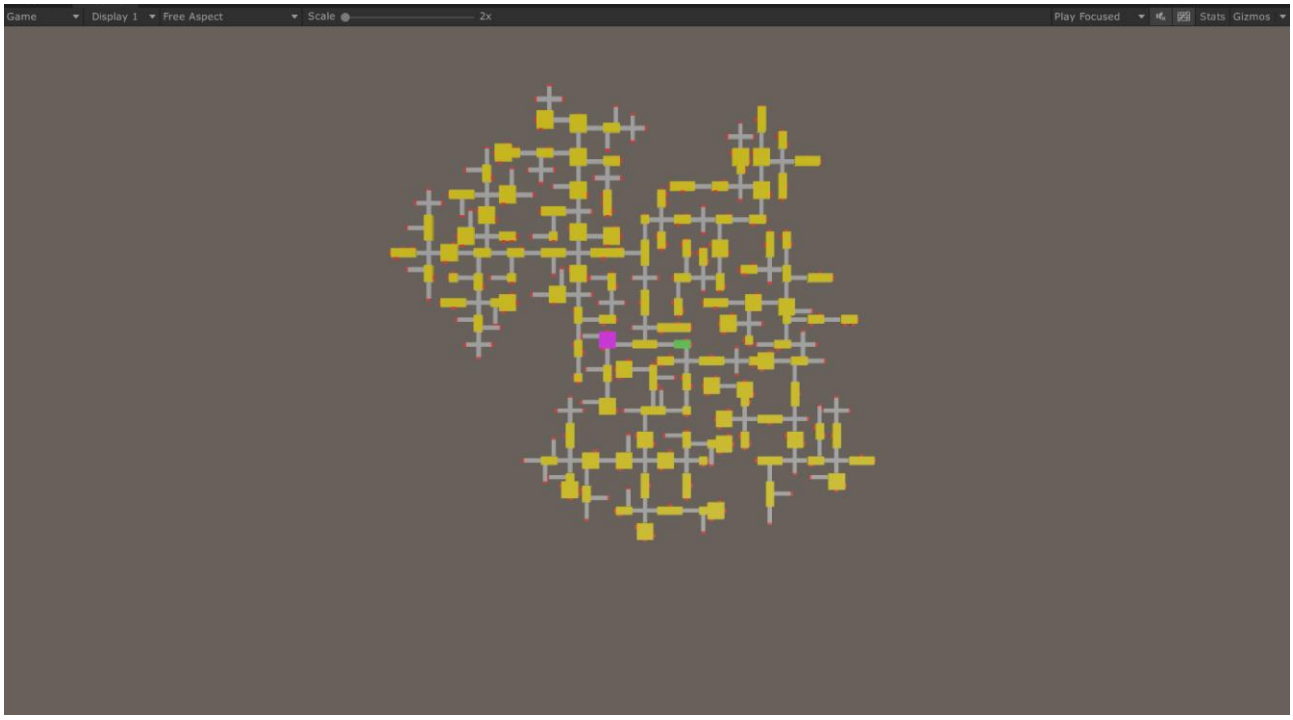

Рис 3.18 Результат варіанта 1

Варіант №3

Параметри:

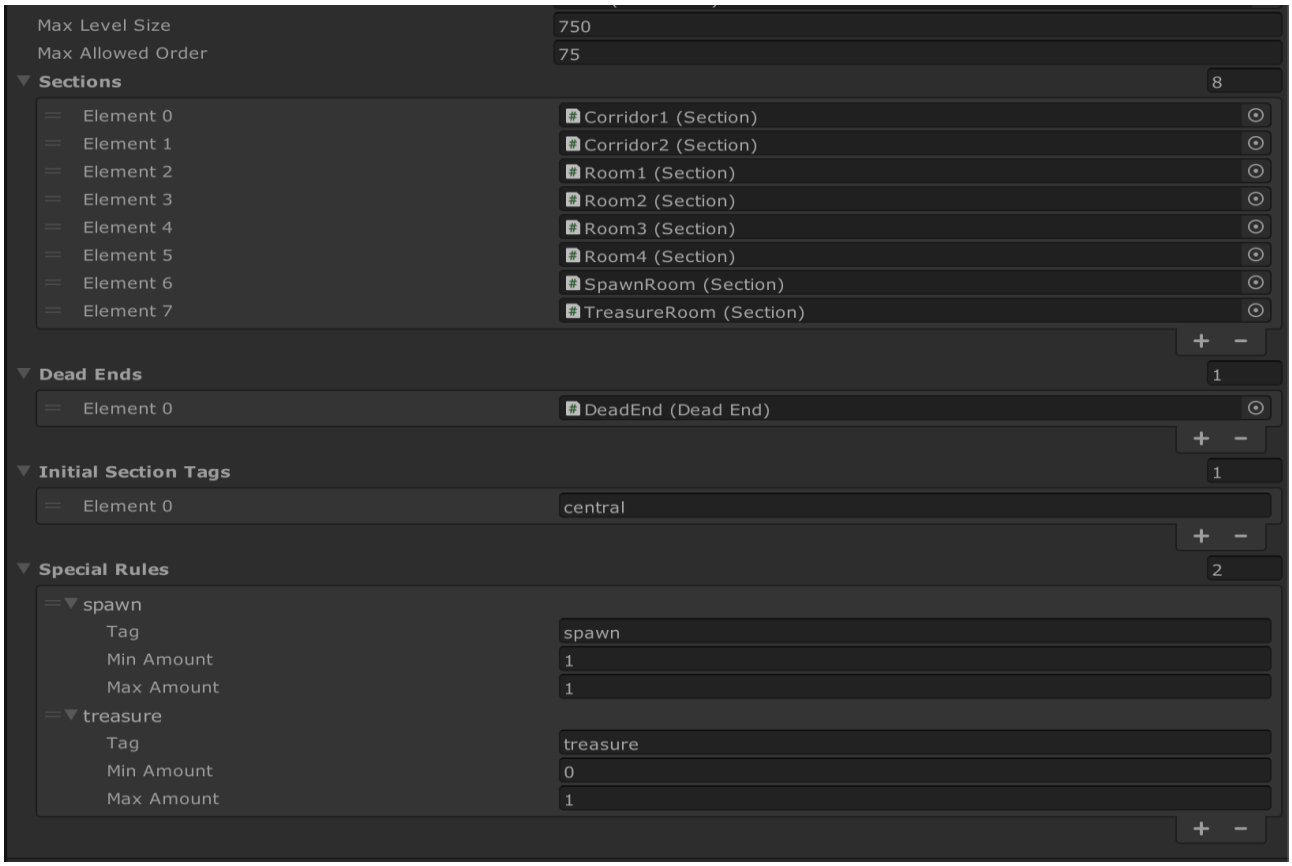

Рис 3.19 Параметри варіанта 3

Результат:

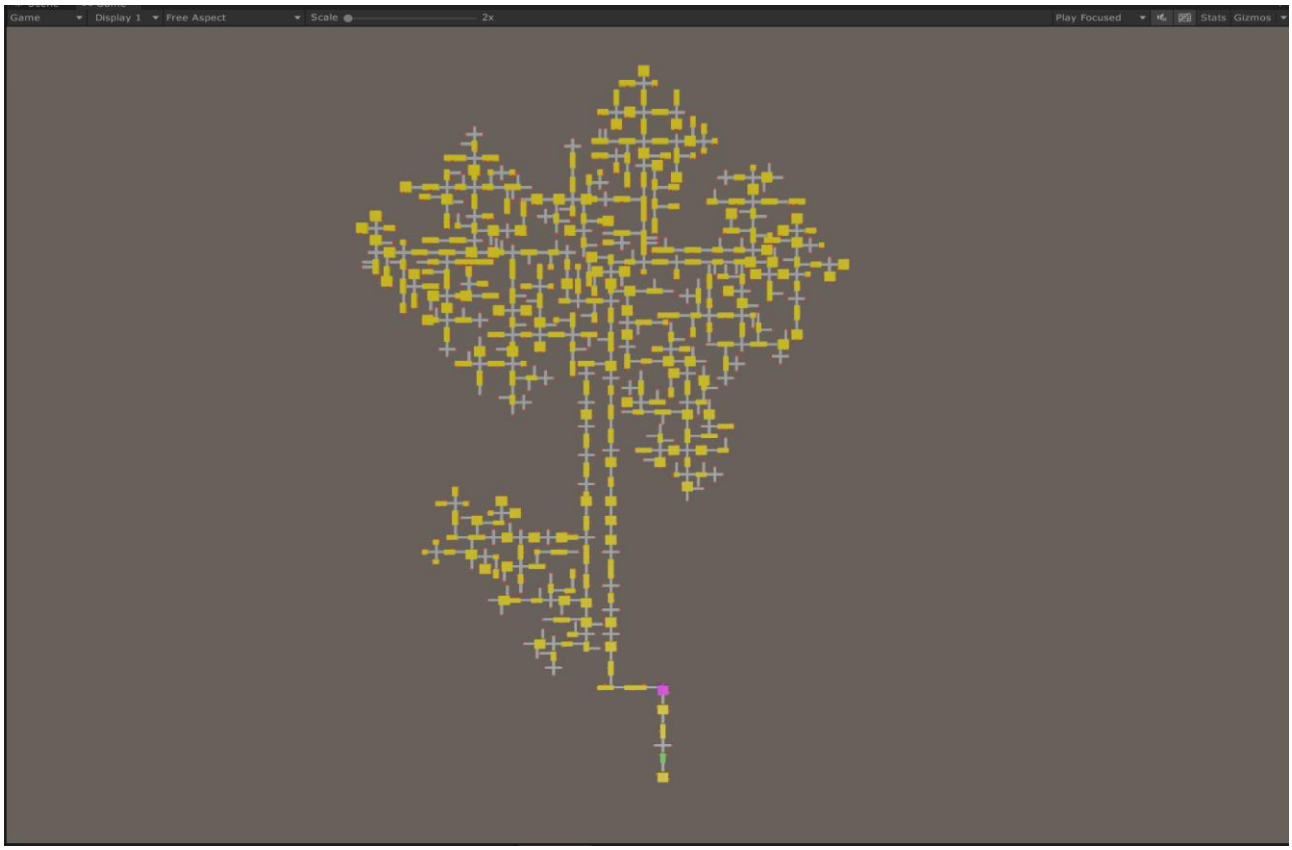

Рис 3.20 Результат варіанта 1

Таким чином програма може генерувати такого роду рівні. Також є можливість модифікувати параметри генерації рівнів під час виконання процедурної генерації, тим самим це дає більш гнучкості програмі для створення чогось нового, більш масштабнішого, не потребуючи для цього як приклад зупиняти виконання програми та зміни параметрів вручну самим програмістом чи геймдизайнером, тим самим скорочуючи час на розробку продукту, тощо.

3.5 Аналіз швидкості процедурної генерації

За приклад будуть взяті дані для процедурної генерації з пункту 3.4. Аналіз швидкості буде проводитись за допомогою вбудованого в C# статичного класу StopWatch. Клас Stopwatch заснований на HPET (High Precision Event Timer, таймер подій високої точності). Цей таймер був введений фірмою Microsoft, щоб раз і назавжди поставити крапку в проблемах часу. Частота цього таймера (мінімум 10 МГц) не змінюється під час роботи системи. Для кожної системи Windows сама визначає, за допомогою яких пристроїв реалізувати цей таймер.

Час для створення маленького рівня складає від 12 до 16 мілісекунд

Час для створення середньої величини рівень складає від 260 до 450 мілісекунд

Час для створення великого рівня складає приблизно 1500 мілісекунд.

ВИСНОВКИ

За результатами проведеної роботи можна зробити такі висновки:

1) розглянуто принципи роботи процедурної генерації рівнів та її методів;
2) проаналізована ефективність та швидкість процедурної генерації рівнів;
3) створена система для процедурної генерації рівнів в ігрових застосунках яка може бути перевикористовувана в інший проектах.

На основі роботи системи, створеної для генерації рівнів та її аналізу ефективності та швидкості роботи, процедурна генерація є доволі ефективна в розробці нового чи вже існуючого проекту

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Alan Thorn — Mastering Unity Scripting, 2015. (Last review 07 January 2021)
2. Alan Thorn — Unity Animation Essentials, 2015. (Last review 07 January 2021)
3. Alan Thorn — How to Cheat in Unity 5: Tips and Tricks for Game Development, 2015. (Last review 07 January 2021)
4. Rouse, Richard. Game Design: Theory & Practice : [англ.]. — 2. — Los Rios Boulevard, Plano, Texas, USA : Wordware Publishing, 2004. — 698 p. — ISBN 1-55622-912-7. (Last review 24 April 2021)
5. Moore, Michael E. Basics of Game Design : [англ.]. — 2. — New York, USA : CRC Press, 2011. — 376 p. — ISBN 13: 978-1-4398-6776-1. (Last review 24 April 2021)
6. Rogers, Scott. Level Up! The Guide to Great Video Game Design : [англ.]. — 2. — USA : Wiley, 2010. — 492 p. — ISBN 978-0-470-68867-0. (Last review 24 April 2021)
7. Schwab, Brian. AI Game Engine Programming : [англ.]. — 2. — Canada : Course Technology, 2009. — 710 p. — ISBN 978-1-5845-0572-3. (Last review 24 April 2021)
8. Kent L., Steven. The Ultimate History Of Video Games : [англ.]. — 1. — New York : Three Rivers Press, 2001. — 608 p. — ISBN 0-7615-3643-4. (Last review 14 May 2021)
9. Joseph Hocking — Unity in Action. Multiplatform game development in C# with Unity 5, 2015. (Last review 14 May 2021)
10. Chris Dickinson — Unity 5 Game Optimization, 2015. (Last visit 14 May 2021)
11. Kenny Lammers — Unity Shaders and Effects Cookbook, 2013. (Last review 14 May 2021)

12. Linowes J.: Unity Projects/ Linowes J.-2015.-286 p. (Last review 14 May 2021)
13. Barrat, James. The latest invention of mankind / James Barrath. - M., 2015. 299 p. (Last review 14 May 2021)
14. Платов, В. Я. Розробка та організація ігрових додатків. Підручник / В.Я. Платов. - М.:, 2015. - 192 с. (Останній перегляд 2 Червня 2021)
15. Любанова, Т.П. Бізнес-план: досвід, проблеми. Зміст бізнес-плану, приклад розробки / Т.П. Любанова, Л.В. Мясоєдова, Т.А. Грамотенко, и др.. - М.: Приор, 2012. - 204 с. 59 (Останній перегляд 2 Червня 2021)
16. Хорхе, Паласиос Unity 5.x. Програмування в іграх. Керівництво / Паласиос Хорхе. - М.: ДМК Пресс, 2017. - 427 с. (Останній перегляд 2 Червня 2021)
17. Алгазинов, Э. К. Аналіз та компютерне моделювання інформаційних процесів та систем / Э.К. Алгазинов, А.А. Сирота. - М.: Диалог-Мифи, 2009. - 416 с. (Останній перегляд 2 Червня 2021)
18. Kim, J. Modeling and Optimization of a Tree Based on Virtual Reality for Immersive Virtual Landscape Generation. Symmetry 2016, 8, 93. (Last review 02 June 2021)
19. Slater, M.; Usoh, M. Simulating peripheral vision in immersive virtual environments. Comput. Graph. 1993, 17, 643–653. (Last visit 02 review 2021)
20. Jeong, K.; Lee, J.; Kim, J. A Study on New Virtual Reality System in Maze Terrain. Int. J. Hum. Comput. Interact. 2018, 34, 129–145. (Last review 02 June 2021)
21. Goldstone W. Unity Game Development Essentials. / W Goldstone. Birmingham: Packt Publishing Ltd., — 2009. — 316 с. (Last review 02 June 2021)
22. Процедурна генерація в гейм-дизайні. Т. Х. Шорт, Т. Адамс
23. Вікіпедія - https://uk.wikipedia.org/wiki

24. Процедурна генерація двовимірного ігрового простору на основі комбінації алгоритму бінарного розбиття простору, алгоритму Крускала та алгоритму «A*» Недодаєв В. А.
25. Куксон Арам, Даулингсока Райан, Крамплер Клинтон Разработка игр на Unreal Engine 4 за 24 часа / Арам Куксон,Райан Даулингсока, Клинтон Крамплер — Бомбора, 2019. - 279 с.
26. Shannon Tom. Unreal Engine 4 for Design Visualization/ Tom Shannon. - Addison-Wesley, 2017. - 195 с.
27. Nixon David. Beginning Unreal Game Development / David Nixon - Apress 2020. - 389 с.
28. Cordone Rachel. Unreal Engine 4 Game Development Quick Start Guide / Rachel Cordone - Packt 2019. - 202 с
29. Шелл Джессі. Геймдизайн. Как создать игру, в которую будут играть все / Шелл Джессі . - Альпіна Паблішер, 2019. - 470 с.
30. Джейсон Шрейер. Кровь, пот и пиксели. Обратная сторона индустрии видеоигр / Шрейер Джейсон. - Форс Україна , 2020. - 368 с.
31. Рэф Костер. Рэф Костер: Разработка игр и теория развлечений / Костер Рэф. - ДМК-Пресс, 2018. - 304 с.
32. Компоненты Движения в Unreal Engine 4 [Електронний ресурс]. Режим доступу: https://www.native-game.com/ue4-docs/ue4-manual/komponenty-dvizhenija-v-unreal-engine-4/.
33. МакКэффри Митч. Unreal Engine VR Cookbook: Developing Virtual Reality with UE4 (Game Design) / Митч МакКэффри. - Бомбора, 2017. - 256 с.
34. Жизан Николя Квантовая случайность. Нелокальность, телепортация и другие квантовые чудеса / Николя Жизан. - Альпина Паблишер, 2016. 355 с.

35. Список_алгоритмів - https://uk.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%96%D0%B2
36. C# in Depth: Fourth Edition 4th Edition. Jon Skeet
37. Game Programming Patterns. Robert Nystrom
38. Design Patterns: Elements of Reusable Object-Oriented Software. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
39. C# 4.0 The Complete Reference, Herbert Schildt
40. Книга Чистий код. Створення і рефакторинг за допомогою Agile, Роберт Сесіл Мартін
41. Книга Грокаємо алгоритми. Ілюстрований посібник для програмістів і допитливих. Адіт’я Бхаргава
42. Unity Game Development Cookbook: Essentials for Every Game 1st Edition. Jon ManningTim NugentParis Buttfield-Addison

Додаток А

Лістинг програми

```
using System.Linq;
using Helpers;
using UnityEngine;

public class Section : MonoBehaviour
{
    public string[] Tags;
    public string[] CreatesTags;
    public Exits Exits;
    public Bounds Bounds;
    public int DeadEndChance;

    private LevelGenerator LevelGenerator;
    private int order;

    public void Initialize(LevelGenerator levelGenerator, int sourceOrder)
    {
        LevelGenerator = levelGenerator;
        transform.SetParent(LevelGenerator.Container);
        LevelGenerator.RegisterNewSection(this);
        order = sourceOrder + 1;

        GenerateAnnexes();
    }

    private void GenerateAnnexes()
    {
        if (CreatesTags.Any())
        {
            foreach (var e in Exits.ExitSpots)
            {
                if (LevelGenerator.LevelSize > 0 && order < LevelGenerator.MaxAllowedOrder)
                    if (RandomService.RollD100(DeadEndChance))
                        PlaceDeadEnd(e);
                    else
```

```
                GenerateSection(e);
            else
                PlaceDeadEnd(e);
        }
    }
}

private void GenerateSection(Transform exit)
{
    var candidate = IsAdvancedExit(exit)
        ? BuildSectionFromExit(exit.GetComponent<AdvancedExit>())
        : BuildSectionFromExit(exit);

    if (LevelGenerator.IsSectionValid(candidate.Bounds, Bounds))
    {
        candidate.Initialize(LevelGenerator, order);
    }
    else
    {
        Destroy(candidate.gameObject);
        PlaceDeadEnd(exit);
    }
}

private void PlaceDeadEnd(Transform exit) => Instantiate(LevelGenerator.DeadEnds.PickOne(),
exit).Initialize(LevelGenerator);
private bool IsAdvancedExit(Transform exit) => exit.GetComponent<AdvancedExit>() != null;
private Section BuildSectionFromExit(Transform exit) =>
Instantiate(LevelGenerator.PickSectionWithTag(CreatesTags), exit).GetComponent<Section>();
private Section BuildSectionFromExit(AdvancedExit exit) =>
Instantiate(LevelGenerator.PickSectionWithTag(exit.CreatesTags),
exit.transform).GetComponent<Section>();
}

using System.Collections.Generic;
using System.Linq;
using UnityEngine;

public class Exits : MonoBehaviour
{
```

```
    public IEnumerable<Transform> ExitSpots => GetComponentsInChildren<Transform>().Where(t => t !=
transform);
}

using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using Exceptions;
using Helpers;
using Structure;
using UnityEngine;
using Debug = UnityEngine.Debug;

public class LevelGenerator : MonoBehaviour
{
    public int Seed;
    public Transform SectionContainer;
    public int MaxLevelSize;
    public int MaxAllowedOrder;
    public Section[] Sections;
    public DeadEnd[] DeadEnds;
    public string[] InitialSectionTags;
    public TagRule[] SpecialRules;
    private List<Section> registeredSections = new List<Section>();
    private Stopwatch stopwatch;
    public int LevelSize { get; private set; }
    public Transform Container => SectionContainer != null ? SectionContainer : transform;

    private IEnumerable<Collider> RegisteredColliders => registeredSections.SelectMany(s =>
s.Bounds.Colliders).Union(DeadEndColliders);
    private List<Collider> DeadEndColliders = new List<Collider>();
    private bool HalfLevelBuilt => registeredSections.Count > LevelSize;

    protected void Start()
    {
        stopwatch = Stopwatch.StartNew();
        Generate();
        stopwatch.Stop();
        Debug.LogError($"time: {stopwatch.ElapsedMilliseconds}");
    }
```

```
    private void Generate()
    {
        if (Seed != 0)
            RandomService.SetSeed(Seed);
        else
            Seed = RandomService.Seed;

        CheckRuleIntegrity();
        LevelSize = MaxLevelSize;
        CreateInitialSection();
        DeactivateBounds();
    }

    private void CheckRuleIntegrity()
    {
        foreach (var ruleTag in SpecialRules.Select(r => r.Tag))
        {
            if (SpecialRules.Count(r => r.Tag.Equals(ruleTag)) > 1)
                throw new InvalidRuleDeclarationException();
        }
    }

    private void CreateInitialSection() => Instantiate(PickSectionWithTag(InitialSectionTags),
transform).Initialize(this, 0);

    public bool IsSectionValid(Bounds newSection, Bounds sectionToIgnore) =>
        !RegisteredColliders.Except(sectionToIgnore.Colliders).Any(c =>
c.bounds.Intersects(newSection.Colliders.First().bounds));

    public void RegisterNewSection(Section newSection)
    {
        registeredSections.Add(newSection);

        if(SpecialRules.Any(r => newSection.Tags.Contains(r.Tag)))
            SpecialRules.First(r => newSection.Tags.Contains(r.Tag)).PlaceRuleSection();

        LevelSize--;
    }
```

```
    public void RegisterNewDeadEnd(IEnumerable<Collider> colliders) =>
DeadEndColliders.AddRange(colliders);

    public Section PickSectionWithTag(string[] tags)
    {
        if (RulesContainTargetTags(tags) && HalfLevelBuilt)
        {
            foreach (var rule in SpecialRules.Where(r => r.NotSatisfied))
            {
                if (tags.Contains(rule.Tag))
                {
                    return Sections.Where(x => x.Tags.Contains(rule.Tag)).PickOne();
                }
            }
        }

        var pickedTag = PickFromExcludedTags(tags);
        return Sections.Where(x => x.Tags.Contains(pickedTag)).PickOne();
    }

    private string PickFromExcludedTags(string[] tags)
    {
        var tagsToExclude = SpecialRules.Where(r => r.Completed).Select(rs => rs.Tag);
        return tags.Except(tagsToExclude).PickOne();
    }

    private bool RulesContainTargetTags(string[] tags) => tags.Intersect(SpecialRules.Where(r =>
r.NotSatisfied).Select(r => r.Tag)).Any();

    private void DeactivateBounds()
    {
        foreach (var c in RegisteredColliders)
            c.enabled = false;
    }
}

using UnityEngine;

public class DeadEnd : MonoBehaviour
{
```

```
    public Bounds Bounds;

    public void Initialize(LevelGenerator levelGenerator)
    {
        transform.SetParent(levelGenerator.Container);
        levelGenerator.RegisterNewDeadEnd(Bounds.Colliders);
    }
}

using UnityEngine;

public class AdvancedExit : MonoBehaviour
{
    public string[] CreatesTags;
}

using UnityEngine;

public class AdvancedExit : MonoBehaviour
{
    public string[] CreatesTags;
}

using System;
using System.Collections.Generic;

namespace Structure
{
    [Serializable]
    public class WeightedSectionList : WeightedList
    {
        public WeightedListItem[] Items;

        public WeightedSectionList()
        {
            keyedItems = new Dictionary<string, WeightedListItem>();
            foreach (var i in Items)
                keyedItems.Add(i.Key, i);
        }
    }
```

```
}

using System;

namespace Structure
{
   [Serializable]
   public class WeightedListItem
   {
      public string Key;
   }
}


using System.Collections.Generic;

namespace Structure
{
   public abstract class WeightedList
   {
      protected Dictionary<string, WeightedListItem> keyedItems;
   }
}


using System;

namespace Structure
{
   [Serializable]
   public class TagRule
   {
      public string Tag;
      public int MinAmount;
      public int MaxAmount;

      private RuleStatus Status => sectionsPlaced < MinAmount
         ? RuleStatus.NotSatisfied
         : sectionsPlaced < MaxAmount
            ? RuleStatus.Satisfied
```

```
                : RuleStatus.Completed;

        public bool Completed => Status == RuleStatus.Completed;

        public bool NotSatisfied => Status == RuleStatus.NotSatisfied;

        private int sectionsPlaced;

        public void PlaceRuleSection() => sectionsPlaced++;
    }

    public enum RuleStatus
    {
        NotSatisfied,
        Satisfied,
        Completed
    }
}
```

Додаток Б

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файла	**Опис**
Пояснювальні документи	
Диплом.doc	Пояснювальна записка кваліфікаційної роботи. Документ Word.
Диплом.pdf	Пояснювальна записка кваліфікаційної роботи в форматі PDF
Програма	
Program.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація.ppt	Презентація до магістерської роботи