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	*Мішина Гліба Денисовича*
	(ПІБ)
академічної групи	*121М-20-1*
	(шифр)
спеціальності	*121 Інженерія програмного забезпечення*
	(код і назва спеціальності)
освітньої програми	*«Інженерія програмного забезпечення»*
	(назва освітньої програми)
на тему:	*Дослідження ефективності застосування методів процедурної генерації для просторового орієнтування на основі грального двигуна Unreal Engine*

_____ *Г.Д. Мішин*

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинговою	інституційною	
розділів кваліфікаційної роботи				
спеціальний	*Проф. Іванченко О.В.*			
економічний	*Проф. Вагонова О.Г.*			
Рецензент	*Проф. Гнатушенко. В.В*			
Нормоконтролер	*Доц. Приходченко С.Д.*			

Дніпро
2022

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

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)
І.М. Удовик
(підпис) (прізвище, ініціали)
« » 20 22 Року

ЗАВДАННЯ
на виконання кваліфікаційної роботи
ступеня магістра

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

студенту *121м-20-1* *Мішину Глібу Денисовичу*
(група) (прізвище та ініціали)

Тема дипломного проекту *Дослідження ефективності застосування методів процедурної генерації для просторового орієнтування на основі грального двигуна Unreal Engine*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 10.12.2021 р. № 1036-с

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – процес просторового орієнтування користувача у віртуальному просторі на базі ігрового двигуна Unreal Engine

Предмет досліджень – моделі та методи процедурної генераціїї для просторового орієнтування користувача у віртуальному просторі на базі ігрового двигуна Unreal Engine

Методи дослідження: неевклідова геометрія, геометрія Лобачевського, ігровий двигун Unreal Engine, розробка програмного забезпечення реалізації методів неевклідової геометрії.

Мета НДР – підвищення рівня когнітивного мислення користувача ігрового програмного забезпечення, за рахунок використання методів неевклідової геометрії у віртуальному просторі.

Вихідні дані для проведення роботи – пакет програмного забезпечення для покращення когнітивного мислення користувача через використання методів та методик ігрового навчання.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Новизна запропонованих рішень визначається тим, що виконано теоретичне обгрунтування та розроблено метод процедурної генерації для реалізації моделі неевклідового простору на базі Unreal Engine.

Практична цінність результатів полягає у тому, що в результаті проведеного дослідження було розроблено програмне забезпечення для покращення процесу просторового орієнтування користувача.

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Для тестування розробленого пакету програмного забезпечення, порівняльного аналізу та використання методів процедурної генерації, розробленого на основі неевклідової моделі простору виконати оцінку ефективності їхнього використання у відповідності з критеріями швидкодії та якості зображення.

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець**
Аналіз теми та постановка задачі	12.09.2021-30.09.2021
Розробка ігрового застунку для створення моделей неевклідової геометрії	01.10.2021-31.10.2021
Реалізація методів процедурної генерації неевклідового простору, тестування отриманих рівнів, оцінка отриманих результатів.	01.11.2021-30.12.2021

6 РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект від реалізації результатів роботи очікується позитивним завдяки скорочення затрат на розробку ігрових додатків для навчання через гру.

Соціальний ефект від реалізації результатів роботи очікується позитивним завдяки покращенню когнітивних навичок користувача, що допоможе йому розвинути навички орієнтації у просторі.

7 ДОДАТКОВІ ВИМОГИ

Завдання видав _____ *Іванченко О.В.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Мішин Г.Д.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *12.09.2021 р.*

Термін подання кваліфікаційної роботи до ЕК *.01.2022*

РЕФЕРАТ

Пояснювальна записка: 95 стор., __ рис., __ таблиці, 3 додатка, 23 джерела.

Об'єкт дослідження: процес просторового орієнтування користувача у віртуальному просторі на базі ігрового двигуна Unreal Engine.

Предмет досліджень – моделі та методи процедурної генераціїї для просторового орієнтування користувача у віртуальному просторі на базі ігрового двигуна Unreal Engine.

Методи дослідження. Для вирішення поставлених задач використані методи: аналізу даних, процедурна генерація, аналіз просторового орієнтування.

Мета роботи – створення ігрового додатку, що допоможе розвинути просторове мислення, та дати розуміння нетипової поведінки ігрового додатку.

Новизна запропонованих рішень визначається тим, що виконано теоретичне обгрунтування та розроблено метод процедурної генерації для реалізації моделі неевклідового простору на базі Unreal Engine.

Практична цінність результатів полягає в тому, що запропоновані в роботі моделі і методи дозволяють використати оптимальний алгоритм процедурної генерації рівня ігровго застосунку, що допоможе покращіти навички у просторовому орієнтуванні на основі ігрового навчання .

Область застосування. Розроблено пакет програмного забезпечення, застосування якого поліпшує когнітивне сприйняття, розвиває креативне мислення та поліпшує навички в просторовому орієнтуванні.

Значення роботи та висновки. Удосконалена методика дозволяє проектувати ігрові додатки напівавтоматично використовуючи процедурну генерацію. Данний ігровий застосунок надає допомогу користувачу для розвитку просторового мислення.

Прогнози щодо розвитку досліджень. Спростити цикл розробки ігрових застосунків, які допомогають користувачу розвивати когнітивні навички.

У розділі «Економіка» проведені розрахунки трудомісткості розробки програмного забезпечення, витрат на створення ПЗ і тривалості його розробки, а також проведені маркетингові дослідження ринку збуту створеного програмного продукту.

Список ключових слів: ігровий двигун, процедурна генерація, комп’ютерна система, проектування, C++, Unreal Engine.

ABSTRACT

Explanatory note: 95 pages, __ figures, __ tables, 3 appendices, 23 sources.

Object of research: the process of spatial orientation of the user in cyberspace based on the game engine Unreal Engine.

The subject of research - models and methods of procedural generation for spatial orientation of the user in cyberspace based on the game engine Unreal Engine.

Research methods. Methods were used to solve the set tasks: data analysis, procedural generation, spatial orientation analysis.

The purpose of the work is to create a game application that will help develop spatial thinking and give an understanding of the atypical behavior of the game application.

The novelty of the proposed solutions is determined by the fact that the theoretical justification and the method of procedural generation for the implementation of the model of non-Euclidean space based on Unreal Engine.

The practical value of the results is that the models and methods proposed in the work allow to use the optimal algorithm of procedural generation of the game application level, which will help to improve skills in spatial orientation based on game learning.

Scope. A software package has been developed, the use of which improves cognitive perception, develops creative thinking and improves skills in spatial orientation.

The value of the work and conclusions. Advanced technique allows you to design game applications semi-automatically using procedural generation. This game application helps the user to develop spatial thinking.

Forecasts for research development. Simplify the development cycle of game applications that help the user develop cognitive skills.

In the section "Economics" calculations of the complexity of software development, the cost of creating software and the duration of its development, as well as marketing research of the market for the software product.List of keywords: game engine, procedural generation, computer system, design, C ++, Unreal Engine.

ЗМІСТ

ВСТУП

Тематикою дипломного проекту є розробка ігрового додатку з використанням неевклідової геометрії, яка допоможе розширити кругозір користувача.

Неевклідова геометрія у ігрових додатках може бути використана для ускладнення орієнтування користувача у просторі. Завдяки чому можна розвивати просторове мислення користувача.

Метою кваліфікаційної роботи є розробка ігрового застосунку, що допоможе користувачу розвинути просторову орієнтацію та когнітивні навички. Використання процедурної генерації спрощує процес створення ігрового додатку. Завдяки процедурній генерації можна досягти створення віртуального простору з використанням неевклідової моделі.

Дана робота дозволяє познайомитись з методами розробок ігрових додатків та розуміння багатовимірних ігрових додатків.

Основною вимогою до даного програмного забезпечення є стабільність роботи. Оскільки Unreal Engine 4 досить вимогливий до графічного процесору, що зумовлено якістю графіки, швидкість роботи додатку може бути не дуже стабільною.

На сьогодні існує невелика кількість аналогічних додатків. Проте вони використовують евклідову геометрію, звичну нам.

Отже задачею даної кваліфікаційної роботи є аналіз методів процедурної генерації для реалізації моделі неевклідової геометрії.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Методологія розробки ігрового застосунку

Розробка ігрового застосунку – це процес написання програмного забезпечення, що допоможе користувачу відволіктись та відпочити від щоденної рутини.

Процес розробки гри зазвичай складається з чотирьох етапів, а саме:

- підготовки;
- уточнення ігрового дизайну;
- виробництво;
- підтримка.

Етапи можуть відрізнятися в залежності від уподобань фірми та специфіки проекту.

1. Підготовчий процес.

Розробка гри на ранніх етапах найчастіше характеризується низькою якістю графіки. Особливо це притаманно різним прототипам ігор.

Перед початком розробки будь-якої гри, команда повинна мати чітко сформульовану ідею та отримати «зелене світло» від видавця/розробника. Якщо розробник і видавець є представниками різних компаній, що частіше зустрічається на практиці, ідею слід схвалити у керівництва та надіслати видавцям на розгляд.

Демонстрація прототипу може допомогти з пошуком партнерів для фінансування, але вона не потрібна авторитетному видавцю. Як тільки буде знайдено зацікавленого видавця, можна розпочати виробництво. Сьогодні ідея гри рідше буває переконливою, якщо в ній не зацікавлений видавець.

Якщо розробник за сумісництвом є видавцем, або це представники підрозділів однієї компанії, схвалення ідеї має надавати лише найвище керівництво. Однак, залежно від розміру видавничої компанії, може знадобитися

кілька спроб, поки ідея не пройде через усі рівні керівництва.

Представником проекту зазвичай є геймдизайнер, але людина з ігрової індустрії на будь-якій іншій посаді також може виконувати цю роль.

Перед початком повного виробництва геймдизайнер повинен надати проектний документ. Це документ, що детально описує концепцію та ігровий процес, може містити деякі попередні ескізи (малюнки) різних аспектів гри. Деякі дизайнери ігор можуть додати приблизний робочий прототип у свій проектний документ, який демонструє один або кілька аспектів гри.

Проектний документ поєднує в собі весь, або більшу частину, початкового концептуального матеріалу. Особливість даного документа полягає в його «жвавість» — фактично він не буде завершений, поки гра не буде в розробці. Він може змінюватися щотижня, іноді щодня. Тому, проектний документ майже ніколи не є повним проектом, навіть якщо буде існувати в певній формі до початку повного виробництва. Хоча він може описати багато аспектів усіх етапів повністю розробленої гри.

Основна команда програмістів і художників можуть починати свою роботу над ідеями, навіть до надходження затвердженого дизайну. Програмісти можуть зайнятися розробкою початкових прототипів для демонстрації однієї або кількох функцій, які деякі посередники бажають побачити в грі. Або вони можуть розробляти фреймворк, який в кінцевому підсумку буде використовуватися в грі. Художники можуть розпочати з малювання ескізів як плацдарм для розробки реальних ігрових ресурсів. Спочатку продюсер може працювати над грою неповний робочий день, але у міру розвитку збільшувати свою зайнятість.

1.2 Особливості використання неевклідової геометрії у віртуальному просторі

Неевклідова геометрія – це будь-яка геометрична система, яка відрізняється від геометрії Евкліда. Раніше неевклідові геометрії були суто

математичними абстракціями, але у 1980-х математик Білл Торнтон революціонізував дослідження 3D-геометрій, уявивши, що подорожує по них. Відтоді математики розпочали створювати анімаційні моделі й навіть симулятори польотів, які дозволяють уявити, як би ви почувалися всередині неевклідового простору.

Порівняно з рендерингом на екрані комп'ютера, VR має перевагу в симуляції того, як світло потрапляє в очі. Коли ви дивитеся на точку в нескінченності в евклідовому просторі, лінії зору обох очей паралельні. У гіперболічному просторі ці лінії розходяться, що викликає зовсім інше сприйняття реальності у спостерігача.

Але насправді це омана. Одним з найдивніших фактів про гіперболічний простір є те, що об'єми та площі в ньому зростають набагато швидше відносно радіуса. Якщо в евклідовому просторі площа зростає пропорційно квадрату радіусу, а об'єм — у кубічному, то в гіперболічному просторі відносини між ними експоненційні.

Це означає, що на таку ж відстань можна втиснути набагато більше місця. Блукаючи по планеті в гіперболічному просторі, спостерігач побачить набагато більше, ніж усе, подолавши таку ж відстань.

1.3 Методи розробки процедурної генерації

Процедурна генерація – це інструментарій послідовного створення даних за допомогою комбінацій алгоритмів, які генерують випадкові послідовності. У комп'ютерній графіці він зазвичай використовується для створення текстур та 3D-моделей. У відеоіграх він використовується для автоматичного створення великої кількості контенту. Залежно від реалізації, переваги процедурної генерації можуть включати менший розмір файлу, більший обсяг контенту та випадковість для певного ігрового процесу.

Існує багато способів створення карт. Жоден із методів не є універсальним рішенням, тому більшість розробників підлаштовують обраний

ними спосіб для своїх ігор. Найбільш популярні способи генерації будуть розглянуті далі.

1. BSP-дерева.

BSP-дерева можна використовувати для створення найпростіших і найбільш характерних для roguelike мап – прямокутних кімнат, з’єднаних коридорами.

Принцип дії такий: якщо в просторі об’єкта провести вертикальну площину, то вона поділить весь об’єктний простір на два півпростору. Якщо продовжити розбиття отриманих напівпросторів далі, то вийде розбиття простору на кластери, всередині яких є один багатокутник або одна грань. Відмінною особливістю цього розбиття є те, що елемент, розташований у напівпросторі, де немає спостерігача, не може перекривати елементи півпростору, в якому перебуває спостерігач. Це дозволяє значно скоротити обсяг пошуку об’єктів, що перекриваються.

2. Алгоритми тунелювання.

Алгоритми тунелювання вкопують коридори та кімнати в тверду землю, як це зробив би справжній архітектор підземелля. За винятком того, що алгоритми часто закінчуються марними або зайвими шляхами.

Високо рандомізований алгоритм тунелювання Drunkard's Walk (з англ. «хода п'яниці») корисний для створення печерних карт, які поєднують відкриті та закриті простори.

3. Клітинні автомати.

Клітинні автомати чудово підходять для розкопування природних печерних систем. На відміну від інших методів у цьому, розробник після генерації карти повинен самостійно надати зв'язки, оскільки деякі алгоритми мають високу ймовірність створення відокремлених областей.

1.4 Існуючі рішення для розробки ігрових застосунків

Хоча ігровий двигун технічно є лише частиною програмного забезпечення, що стоїть за відео грою, це більше, ніж нескінченний рядок коду. Хороший двигун — це серце й мозок, які роблять вашу улюблену гру можливою, дозволяючи розробникам втілювати певні ідеї в життяв життя. Все це дозволяє користувачам розвинути межі існуючого та навіть дозволяє їм мандрувати в інших світах, які вони собі увляють. Існує декілька вже готових варіантів ігрових двигунів:

1. CryEngine.
2. Lumberyard.
3. Phaser.
4. Unreal Engine і Unity.

Для того щоб вибрати один з існуючих, треба розглянути кожен з них більш детально.

1. CryEngine

CryEngine — ігровий двигун, створений німецькою приватною компанією Crytek у 2002 році і спочатку використовуваний у шутері від першої особи Far Cry. CryEngine — це комерційний двигун, який пропонується для ліцензування іншим компаніям. Станом на 30 березня 2006 року Ubisoft зберігає всі права на двигун.

Двигун був ліцензований NCSoft для розробки MMORPG Aion: Tower of Eternity.

Наприкінці вересня 2009 року брати Ерлі, засновники Crytek, дали інтерв'ю британському журналу Develop, в якому вони заявили, що спочатку CryEngine не планувалося ліцензувати третьою стороною. CryEngine планувалося бути закритим двигуном для суто внутрішнього використання.

Плюси:

- код двигуна можна змінювати, що приносить і радість, і біль;
- існує магазин вже готових ассєтів;

- має підтримку консолей та персональних комп’ютерів водночас;

Мінуси:

- досить вимогливий до розрахункової потужності;
- не розгорнута документація, що може бути ускладненням для розробки з нуля.

2. Lumberyard

Amazon Lumberyard — це безкоштовний кросплатформний ігровий двигун AAA, розроблений Amazon і заснований на архітектурі движка CryEngine. Двигун має інтеграцію з веб-сервісами Amazon, що дозволяє розробникам створювати або розміщувати свої ігри на серверах Amazon, а також транслювати відео через Twitch. Вихідний код доступний для кінцевих користувачів, але є обмеження щодо використання: заборонено публікувати вихідний код движка або використовувати його для випуску власного ігрового движка. Lumberyard було запущено 9 лютого 2016 року разом із GameLift, керованою службою для розгортання та розміщення багатокористувацьких ігор на основі комісій, що дозволяє розробникам легко розробляти ігри, які залучають «великі та активні шанувальники». Наразі програмне забезпечення знаходиться на стадії бета-тестування, і його можна використовувати для створення ігор для Windows, Xbox One та PlayStation 4.

Плюси:

- кросплатформність;
- доступність;
- докладна документація;

Мінуси:

- досить сирий продукт, на даний момент;
- платна інтеграція сервісів Amazon.

3. Phaser

Phaser — це 2D-ігровий фреймворк, який використовується для створення ігор HTML5 для настільних і мобільних пристроїв. Це безкоштовне програмне забезпечення, розроблене Photon Storm.

Phaser внутрішньо використовує засоби візуалізації Canvas і WebGL і може автоматично перемикатися між ними залежно від підтримки браузера. Це дозволяє швидко відтворювати на настільних комп'ютерах і мобільних пристроях. Для візуалізації використовується бібліотека Pixi.js.

Ігри можна компілювати для iOS, Android і настільних програм для настільних комп'ютерів за допомогою сторонніх інструментів, таких як Apache Cordova і phonegap.

З огляду на це, ви можете загорнути свою гру у власну програму за допомогою таких інструментів, як Cordova і Phonegap, сама гра ніколи не компілюється. "Гра" просто запускається як JavaScript у відповідному браузері. Це означає, що продуктивність не схожа на нативну компільовану програму.

Плюси:

- ігровий додаток може працювати на будь-якому пристрої який має додаток браузера;
- чудова документація;
- низький рівень входу для розробника;

Мінуси

- відсутність розробляти багатовимірну гру;
- низька продуктивність при використанні шейдерів;
- однопоточна робота коду, за рахунок JavaScript;
- низька безпека коду.

Порівняння ігрових двигунів Unreal Engine і Unity3d**.**

Unity3d і Unreal Engine 4 - це одні з найпопулярніших ігрових движків, доступних на сьогоднішній день. Хоча обидва - відмінні ігрові движки, в залежності від того, що ви збираєтеся зробити, кожен може стати для вас оптимальним варіантом. Можливо, ви вважаєте за краще простий зручний інтерфейс, або, може бути, мова програмування для вас більш пріоритетний - незалежно від тих чи інших обставин, давайте подивимося на відмінності, а також на сильні і слабкі сторони кожного движка. Тоді ви і зможете вирішити, який з них буде відповідати вашим потребам.

По перше, треба визначитись, для чого потрібно використовувати ігровий двигун. На багатьох ігрових двигунах можна розробити 2d платформер або 3d шутер від першої особи. Також можна створити деякий гібрид 2d і 3d. Для початку можна почати з першої невеликої гри і розробити простий пазл з базової фізикою. Також можна реалізувати запуск гри в веб-браузері, або на мобільних платформах. Досить важливим є етап вирішення монетизації гри, або як вона буде приносити грошові кошти її розробнику. Є декілька способів заробітку на іграх, внутрішні покупки, реклама або просто зробити гру платною.

Визначившись з цими етапами, можуть допомогти визначити, який движок слід використовувати. Обидва движка можуть впоратися з будь-якою з цих завдань, але в залежності від того, що ви робите, один движок може бути набагато зручніше і оптимальніше.

Якщо ви хочете робити мобільні ігри - Unity буде ідеальним рішенням. Це підтверджується домінуванням Unity серед розробників мобільних ігор, а також великою кількістю плагінів для використання нативних можливостей мобільних платформ: реклама, внутрішні покупки, аналітика, ігрові центри і т. Д. - все це інтегрується в гру за лічені хвилини. Якщо ви націлені на розробку 2d гри, Unity теж буде прекрасним вибором, тому що саме у нього є прекрасні можливості для створення 2d ігор. Хоча Unreal Engine 4 останнім часом намагається заманити розробників мобільних додатків, обіцяючи не менші, а навіть великі можливості для 2d ігор.

Якщо ви хочете створювати 3d ігри, Unity також дуже потужний інструмент для розробки 3d ігор. Хоча графічно він далеко не на тому рівні, що Unreal Engine 4. При необхідності використовувати next-gen графіком кращим рішенням буде Unreal Engine 4.

В Unreal Engine 4 використовується мова програмування C ++. У Unity3d в основному C # або JavaScript. Яка програма краще з точки зору мов програмування - дійсно зводиться до особистих вподобань. Деякі люди думають, що C ++ є архаїчним, а інші моляться на нього. Якщо ви віддаєте перевагу якійсь із цих мов, то вибір движка для вас може бути досить очевидним.

Але Unreal Engine 4 має рішення для людей, які бояться високого порогу входження в C ++. Це Blueprint - редактор візуального скриптинга. Технічно вам не потрібно писати жодного рядка коду. Це дуже зручно для створення швидких прототипів, і ви навіть можете створювати цілі гри за допомогою Blueprint . Якщо ви не майстер програмування , Unreal Engine 4 буде для вас відмінним варіантом. До слова, в Unity є схоже рішення. У Unity Asset Store можна придбати розширення для редактора під назвою Playmaker, яке так само дозволяє розробляти прототипи ігор без написання коду.

Asset Store

Обидва движка Unreal Engine 4 і Unity3d мають свій магазин Ассет: з них можна завантажити готові 3d моделі персонажів і оточення, текстури і навіть такі речі, як звуки і системи частинок. Проте, Unity3d однозначно виходить на перше місце з точки зору кількості Ассет в магазині. У ньому є все - від анімацій і генераторів GUI до розширень редактора для управління штучного інтелекту. Там є все, що потрібно для створення гри.

1.5 Основи неевклідової геометрії

Як зазначено вище, неевклідова геометрія це будь-яка геометрія, що відрізняється від геометрії Евкліда. На сьогоднішній день основними неевклідовими моделями геометрії вважаються сферична геометрія та геометрія Лобачевського. Щоб принципи цих моделей були більш зрозумілі, пропонуємо порівняти основні математичні формули цих геометрій.

Основні положення евклідової геометрії базуються на застосуванні теореми Піфагора, математичне висловлювання якої записується у вигляді [1]

$$ds^2 = dx^2 + dy^2, \tag{1.1}$$

де ds – довжина гіпотенузи,

dx і dy – довжини катетів прямокутного трикутника.

Сферична геометрія**:**

$$ds^2 = dx^2 + \cos^2(y/R)dy^2, \tag{1.2}$$

де R – радіус сфери,

ds – довжина гіпотенузи,

dx і dy – довжини катетів прямокутного трикутника.

Геометрія Лобачевського**:**

$$ds^2 = dx^2 + ch^2(y/R)dy^2, \tag{1.3}$$

де ds – довжина гіпотенузи,

dx і dy – довжини катетів прямокутного трикутника,

R - радіус кривизни площини Лобачевського,

ch - гіперболічний косинус.

Отже, порівнюючи математичні формули, стає зрозуміло, що основою для неевклідових моделей геометрії є робота у просторі, а не на площині.

Сферична геометрія — це розділ геометрії, у якому вивчаються геометричні фігури на поверхні кулі. Вона виникла ще в давнину у зв'язку можливістю людей користуватися географією та астрономією. На рисунку 1.1 зображено класичний трикутник у сферичний геометрії.

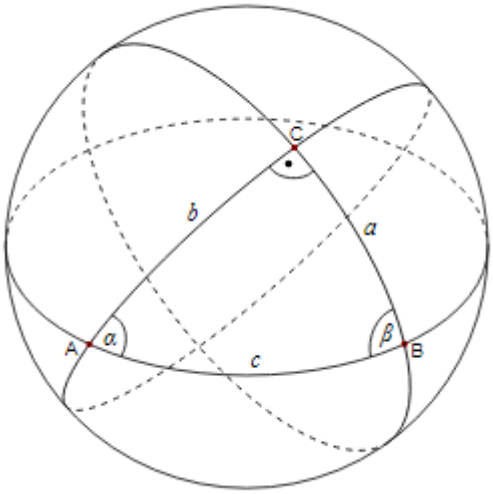

Рис. 1.1. Сферичний трикутник

Слід виділити основні поняття, що використовуються у сферичній геометрії та геометрії Лобачевського.

Сферична геометрія [6]:

1. Велике коло — це коло, яке ділить кулю (сферу) на дві рівні половини. Центр великого кола завжди збігається з центром кулі. Для кращого розуміння розглянемо на прикладі глобусу. Усі меридіани є великими колами, але, якщо дивитися з паралелей, лише екватор є великим колом. Усі інші паралелі — маленькі кола.
2. Великі кола на поверхні кулі грають роль, подібну до прямих ліній у планіметрії. Найкоротший шлях між будь-якими двома точками проходить уздовж лінії великого кола.
3. Через будь-які дві точки на поверхні кулі, за винятком діаметрально протилежних точок, можна провести одне велике коло. Через

діаметрально протилежні точки на кулі можна провести будь-яку кількість великих кіл.

4. Будь-які два великих кола перетинаються по прямій, що проходить через центр сфери, а кола великих кіл перетинаються в двох діаметрально протилежних точках.

Геометрія Лобачевського:

Аксіома Лобачевського є точним запереченням аксіоми Евкліда (якщо всі інші аксіоми виконуються).

У випадку, коли жодна пряма, що лежить із даною прямою в одній площині і не перетинає її, не проходить через точку, яка не лежить на даній прямій, виключається в силу інших аксіом (аксіом абсолютної геометрії).

Так, наприклад, сферична геометрія і геометрія Рімана, в якій будь-які дві прямі перетинаються, а отже, не виконуються ні аксіома евклідової паралельності, ні аксіома Лобачевського, несумісні з абсолютною геометрією.

Основним твердженням цієї геометрії є заперечення аксіоми Евкліда про паралельні прямі.

Замість неї використовується наступна аксіома:

Через точку, яка не лежить на даній прямій, проходять принаймні дві прямі, які лежать з цією прямою в одній площині і не перетинають її [4].

1.5.1 Евклідовий простір

Евклідів простір у первісному значені — це простір, властивості якого описуються аксіомами евклідової геометрії. У цьому випадку передбачається, що простір має розмірність, рівну 3, тобто він тривимірний.

В більш загальному сенсі, він може позначати один із подібних і тісно пов’язаних об’єктів: скінченновимірний дійсний векторний простір із введеним на нього додатним певним скалярним добутком; або метричний простір, що відповідає такому векторному простору. За вихідне буде взято перше визначення [7].

n-вимірний евклідів простір зазвичай позначають як $\mathbb{E}^n$. Якщо з контексту ясно, що простір обладнано природною евклідовою структурою, тоді позначення також використовується.

Коли потрібно визначити евклідовий простір, найпростішим варіантом буде взяти за основу поняття крапкового добутку.

Евклідів векторний простір визначається як скінченновимірний векторний простір над полем дійсних чисел, на парах векторів якого задана дійсна функція з такими трьома властивостями [1]:

• білійність:

для будь-яких векторів u, v, wі та для будь-яких дійсних чисел α, b справедливі відношення:

$$(au + bv, w) = a(u, w) + b(v, w) \text{ і } (u, av + bw) = a(u, v) + b(u, w). \quad (1.4)$$

• симетрія:

для будь-яких векторів u, v справедлива рівність $(u, v) = (v, u)$.

• позитивна визначеність $(u, u) \geq 0$:

для будь-якого u, при цьому $(u, u) = 0 \Rightarrow u = 0$.

Афінний простір, що відповідає такому векторному простору, називається евклідовим афінним простором або просто евклідовим простором [1].

Прикладом евклідового простору є координатний простір $\mathbb{R}^n$, що складається з усіх можливих наборів дійсних чисел $(x\_1, x\_2 \ldots, n)$, скалярний добуток де визначається за формулою

$$(x, y) = \sum_{i=1}^{n} x_i y_i = x_1 y_1 + x_2 y_2 + \ldots + x_n y_n. \quad (1.5)$$

1.5.2 Метрика неевклідової геометрії на площині

Моделі геометрії Лобачевського підтвердили її несуперечливість, а точніше, показали, що геометрія Лобачевського така ж послідовна, як і геометрія Евкліда.

Сам Лобачевський дав основи своєї аналітичної геометрії, і таким чином він фактично окреслив таку модель. Він також зауважив, що орісфера в просторі Лобачевського ізометрічни евклідової площини, тим самим фактично запропонував зворотну модель. Тим не менш, сама концепція моделі була уточнена в роботах Бельтрамі та інших [4].

Проективна модель є основною метрикою неевклідової геометрії на площині. Вперше модель площини Лобачевського була запропанована Бельтрамі.

Площина — це внутрішня частина кола, лінія — це хорда кола без кінців, а точка — точка всередині кола. Рухом називається будь-яке перетворення кола в себе, що перетворює хорди в акорди. Відповідно фігури всередині кола, які перекладаються одна в іншу в результаті таких перетворень, називають рівними.

Тоді виявляється, що будь-який геометричний факт, описаний такою мовою, являє собою теорему або аксіому геометрії Лобачевського.

Іншими словами, будь-яке твердження про геометрію Лобачевського на площині є не що інше, як твердження евклідової геометрії, що стосується фігур всередині кола, лише переказаних у зазначених термінах [2].

На рисунку 1.2 зображено що евклідова паралельна аксіома явно не виконується, оскільки через точку P, яка не лежить на даній хорді a, проходить стільки ж хорд (прямих), які її не перетинають.

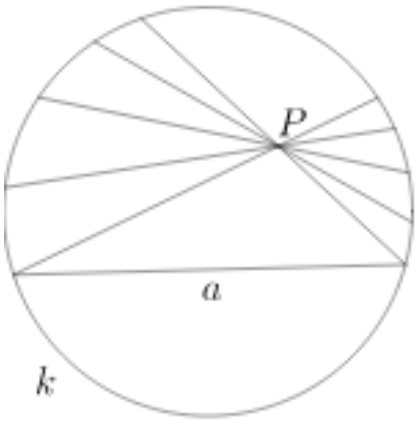

Рис. 1.2. Проективна модель

1.6. Геометрія Лобачевського

Лобачевський на основі геометричних понять і своєї аксіоми будував свою геометрію і доводив теореми геометричним методом, подібно до того, як це робиться в геометрії Евкліда.

Різниця між геометрією Лобачевського і геометрією Евкліда починається з теорії паралельних прямих, яка саме й послужила основою. Усі теореми, незалежні від паралельної аксіоми, є спільними для обох геометрій.

Вони утворюють так звану абсолютну геометрію, до якої, наприклад, належать ознаки рівності трикутників. Такі розділи як тригонометрія та початки аналітичної та диференціальної геометрії були побудовані, дотримуючись саме теорії паралелей.

Для прикладу, наведемо декілька фактів геометрії Лобачевського, які відрізняють її від геометрії Евкліда і встановлені самим Лобачевський.

Через точку P, яка не лежить на даній прямій R, проходить нескінченно багато прямих, які не перетинають R і знаходяться з нею в одній площині; серед них є два крайніх x, y, які називають асимптотично паралельними (іноді просто паралельними) прямими R, а інші називаються ультрапаралельними [6].

Коли точка P віддаляється від прямої, кут θ між перпендикуляром PB від P до R і кожним з асимптотично паралельних (званих кутом паралельності)

зменшується з 90° до 0° (у моделі Пуанкаре кути в звичайний сенс збігаються з кутами в сенсі Лобачевського, і тому на цьому факті можна дивитися безпосередньо).

Паралель x з одного боку (і y протилежного боку) асимптотично наближається до a, а з іншого - нескінченно віддаляється від нього (у моделях відстані важко визначити, а тому цей факт безпосередньо не видно).

Для точки, розташованої від даної прямої на відстані РВ = а, Лобачевський дав формулу для паралельного кута Р (а):

$$\Theta = \prod(a) = 2arctge^{-\frac{a}{q}}, \tag{1.6}$$

де q — деяка константа, пов'язана з кривизною простору Лобачевського [4].

Він може служити абсолютною одиницею довжини так само, як у сферичній геометрії особливе положення займає радіус кулі.

Якщо прямі мають спільний перпендикуляр, тоді вони вважаються ультрапаралельними, тобто нескінченно розходяться від нього в обидва боки. До будь-якого з них можна відновити перпендикуляри, які не доходять до іншої прямої.

У геометрії Лобачевського немає подібних, але нерівних трикутників; трикутники рівні, якщо їхні кути рівні.

Сума кутів будь-якого трикутника менша π і може бути як завгодно близькою до нуля. Різниця між 180 ° і сумою кутів трикутника АВС в геометрії Лобачевського додатна та називається дефектом трикутника. Це можна побачити безпосередньо в моделі Пуанкаре. Різниця

$$\delta = \pi - (\alpha + \beta + \gamma), \tag{1.7}$$

де α, β, γ – кути трикутника, пропорційна його площині:

$$S = q^2 \cdot \delta . \tag{1.8}$$

Ви можете побачити з формули (2.1), що існує максимальна площа трикутника, а це скінченне число: πq^2.

Рівновіддаленою лінією, або гіперциклом, називається рівна відстань від прямої, що є не прямою, а особливою кривою.

Граничним колом, або хороциклом, називається межа кіл нескінченно зростаючого радіусу, що є не прямою лінією, а особлива крива.

Межею сфер нескінченно зростаючого радіусу є не площина, а особлива поверхня - гранична сфера, або оросфера.

Примітно, що на ньому має місце евклідова геометрія. Це послужило основою для виведення Лобачевського формул тригонометрії.

1.6.1 Моделі геометрії Лобачевського

Існує декілька основних моделей геометрії Лобачевського:

1. Конформо-евклідова модель, або модель Пункаре.
2. Модель на гіперболоіді.
3. Поверхня постійної негативної кривизни.

1. Конформо-евклідова модель, або модель Пункаре.

На рисунку 1.3 зображено, що за площину Лобачевського береться внутрішня частина кола, дуги кіл, перпендикулярні до кола цього кола. Його діаметри вважаються прямими, а перетворення вважаються рухами, отримані за допомоги комбінацій зворотів відносно кіл, дуги яких служать як прямі.

Модель Пуанкаре чудова тим, що кути зображені в ній, вважаються звичайними кутами.

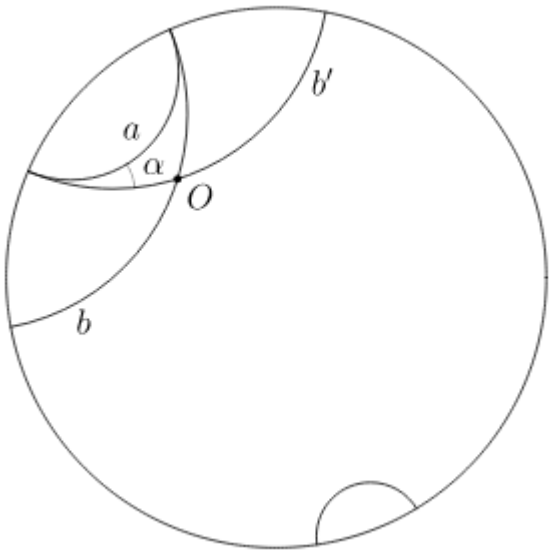

Рис. 1.3. Конформа-евклідова модель

2. Модель на гіперболоїді

У просторі підпису (+, +, -) розглянемо дволистовий гіперболоїд

$$x^2 + y^2 - t^2 = -1. \tag{1.9}$$

Виберемо верхню складову t > 0. Зауважимо, що ця компонента є просторовою. Зокрема, квадратична формула (1.8) визначає на ній метрику; при цій метриці верхній компонент є моделлю площини Лобачевського.

Прямі (інакше кажучи, геодезичні) в цій моделі є перерізами гіперболоїда площинами, що проходять через початок координат.

Перспективна проекція на горизонтальну площину з центром у початку координат перетворює цю модель у проективну модель. Перспективна проекція на горизонтальну площину з центром (0,0, -1) перетворює цю модель на конформну евклідову [4].

3. Поверхня постійної негативної кривизни

Інше аналітичне визначення геометрії Лобачевського полягає в тому, що геометрія Лобачевського визначається як геометрія рімановового простору постійної негативної кривизни. Це визначення фактично було дано ще в 1854 р.

Ріманом і включало модель геометрії Лобачевського як геометрію на поверхнях постійної кривизни. Проте Ріман прямо не пов’язував свої конструкції з геометрією Лобачевського, а його доповідь, в якій він їх повідомляв, не була зрозуміла і була опублікована лише після його смерті [5].

1.7. Основи орієнтування в просторі для віртуального простору

Для орієнтування в просторі мозок людини створює проекцію у виді гексагональної решітки.

У 1971 році О'Кіфу вдалося зробити перший і найважливіший крок у цьому дослідженні, за яке тепер він був удостоєний Нобелівської премії. Вчений експериментував з щурами і виявив, що певні клітини в мозку тварин завжди активні, коли тварини знаходяться безпосередньо в певному місці.

Ці «просторові клітини» розташовані в гіпокампі, тобто з точки зору еволюції - дуже старій області мозку, в якій зливається вся інформація від органів чуття. Потім вона спрямовується далі в кору головного мозку. Просторові клітини не контролюються безпосередньо зовнішніми сенсорними впливами, вони існують як самостійні, утворюють «внутрішню карту».

Як людина орієнтується в просторі - спогади про місце перебування зберігаються завдяки специфічній активності в цих клітинах гіпокампу.

Мей-Брітт і Едвард Мозер представили результати дослідження навігаційної системи мозку. Обидва вчені, які, до речі, є п'ятими подружжями, які отримали Нобелівську премію, вивчали на щурах, як поводяться нервові повідомлення в гіпокампі, коли тварина рухається. Вони виявили несподіване: у сусідній області мозку, так званій енторинальній корі, клітини активуються у вигляді гексагональної сітки.

Кожен нейрон сітки (або нейрони сітки) діяв специфічно, коли щури потрапляли в певні місця. Такі нейронні мережі є системою координат в голові. Разом з рештою клітин енторинальної кори, які реєструють положення голови та

розміщення стінок у просторі, просторові клітини оксифа утворюють шаблон, який робить можливим реальну орієнтацію в просторі.

Класичні багатовимірні ігрові додатки використовують образи об’єктів звичниі для людини, дома, кімнати. Всі ці проекції вже існують у пам’яті користувача, та не складає складнощів для орієнтування. На приклад, ігровий персонаж знаходиться у відкритій місцевості, користувач бачить перед собою старинні руїни та сходинки (Рис. 1.4).

Рис. 1.4. Приклад орієнтування у віртуальному просторі

Завдяки пам’яті проекції користувач розуміє, що ці сходи приведуть його вгору, де він зможе проаналізувати місцевість. Таким чином інформація яку ми запам’ятовуємо у реальному просторі можна використовувати у віртуальному и навпаки (Рис. 1.5).

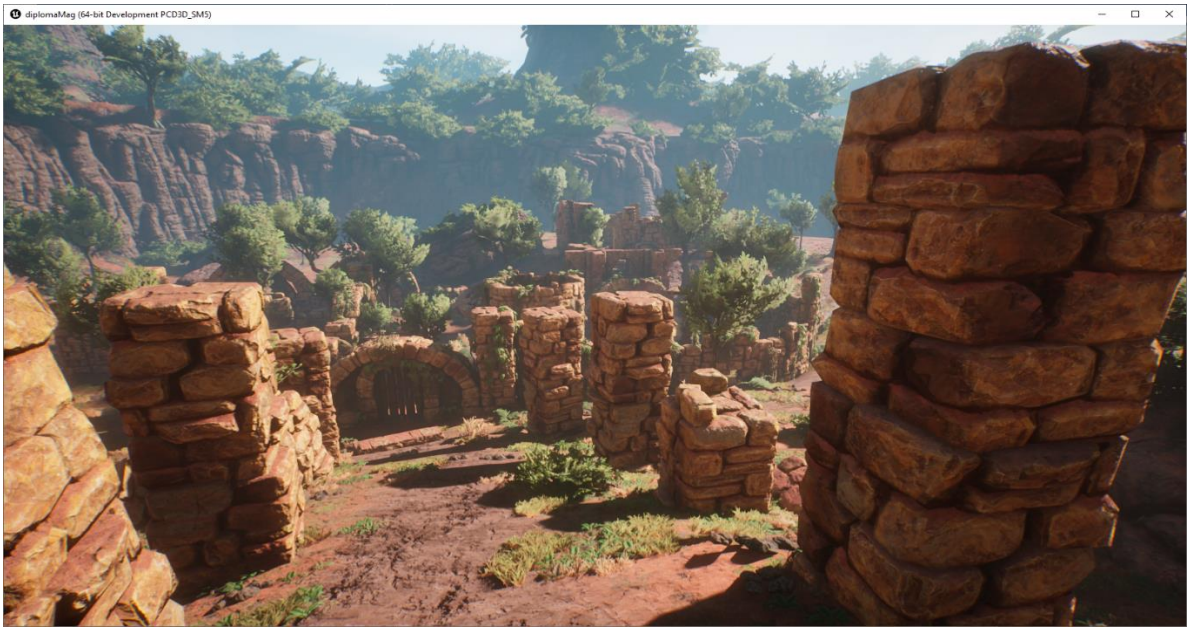

Рис. 1.5. Приклад використання інформації реального простору у віртуальному

Таким чином орієнтування у віртуальному просторі базується на орієнтуванні у реальному житті. Вся інформація що була здобута у житті сумісна з віртуальним простором.

1.7.1 Використання псевдосфери для ускладнення сприйняття віртуального простору

Псевдосфера це одна з моделей геометрії Лобачевського.

Італійський математик Еудженіо Бельтрамі в 1868 році помітив, що геометрія на шматку площини Лобачевського збігається з геометрією на поверхнях постійної негативної кривизни, найпростішим прикладом якої є псевдосфера. Якщо точки і прямі на кінцевому відрізку площини Лобачевського пов’язані з точками і найкоротшими лініями (геодезичними) на псевдосфері, а рух у площині Лобачевського порівняти з рухом фігури вздовж псевдосфери з вигином, тобто деформації, що зберігає довжину, то будь-яка теорема геометрії Лобачевського буде відповідати тому, що на псевдосфері. У цьому випадку довжини, кути, площі розуміють у сенсі їх природного вимірювання на псевдосфері.

Однак тут дається лише локальна інтерпретація геометрії, тобто на обмеженій ділянці, а не на всій площині Лобачевського. Подібну модель дає поверхня Діні - це ізометричне занурення області площини Лобачевського, обмеженої гороциклом (Рис. 1.6).

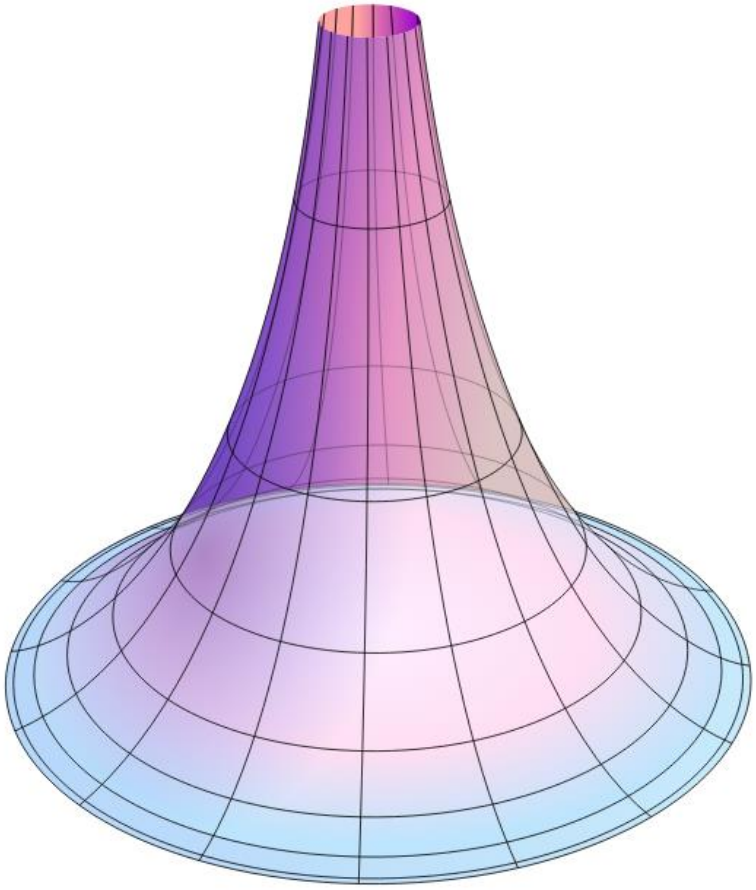

Рис. 1.6. Псевдосфера геометрії Лобачевського

1.8. Застосування неевклідової геометрії у віртуальному просторі

Неевклідова геометрія застосовується у багатьох ігрових додатках. Її застосування частіш за все зумовлено чимось новим та не звичним для користувача.

Ідеї неможливої архітектури, спотвореного простору та нетипової навігації — чудові ідеї для ігор-головоломок. Це жанр, у якому неевклідова геометрія показала себе у всій красі.

Однією з найвідоміших неможливих космічних ігор була Antichamber. Це філософська гра-головоломка від першої особи, в якій гравець послідовно розгадує головоломки і пробирається до кінцевої локації (Рис. 1.7).

Знайомство з неевклідовим простором відбувається на самому початку гри. Немає роликів, які б пояснювали, що відбувається, тому гравцеві доведеться розібратися в цьому самостійно.

Однією з перших перешкод у грі є дві драбини - одна веде вгору, а інша вниз. Який би шлях не вибрав гравець, він неминуче телепортується в коридор, який привів його до розвилки. Щоб вийти з цієї просторової петлі, він повинен розвернутися і піти в протилежному напрямку. Тільки тоді йому буде відкритий шлях на наступний рівень.

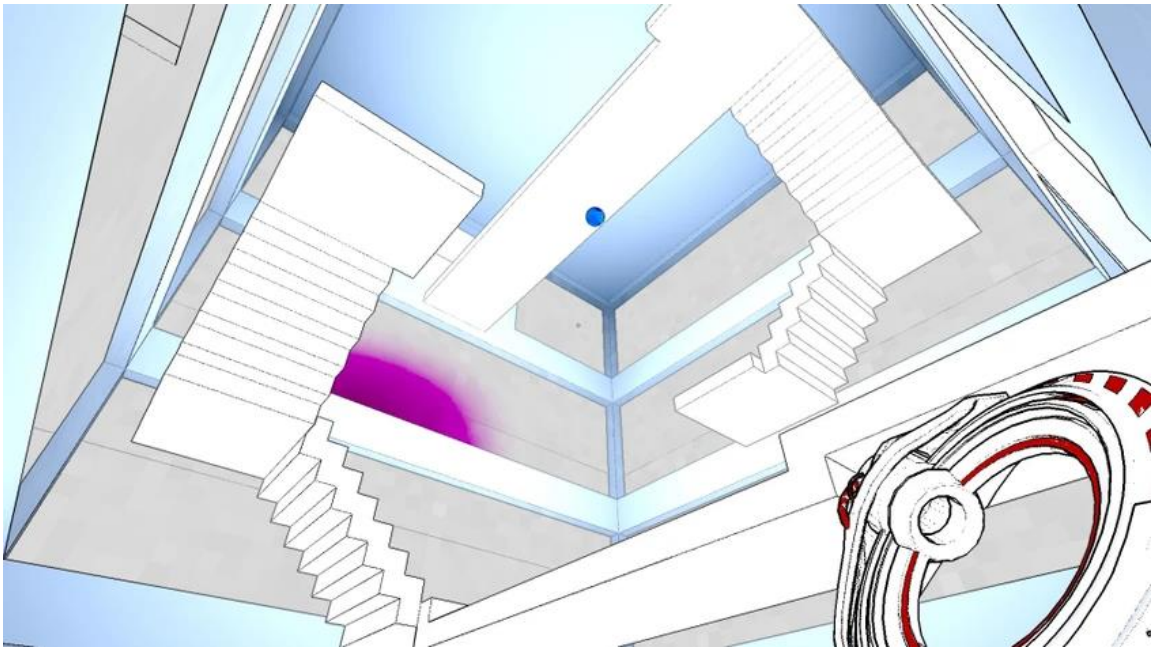

Рис. 1.7. Кадр із Antichamber

Також яскравим представником використання неевклідової геометрії є Manifold Garden(Рис. 1.8).

Творець Manifold Garden Вільям Чир вирішив піти далі і зробити неможливі простори ядром ігрового процесу. Це дає гравцеві можливість

маніпулювати силою тяжіння та досліджувати архітектурні споруди, що нагадують літографії Ешера.

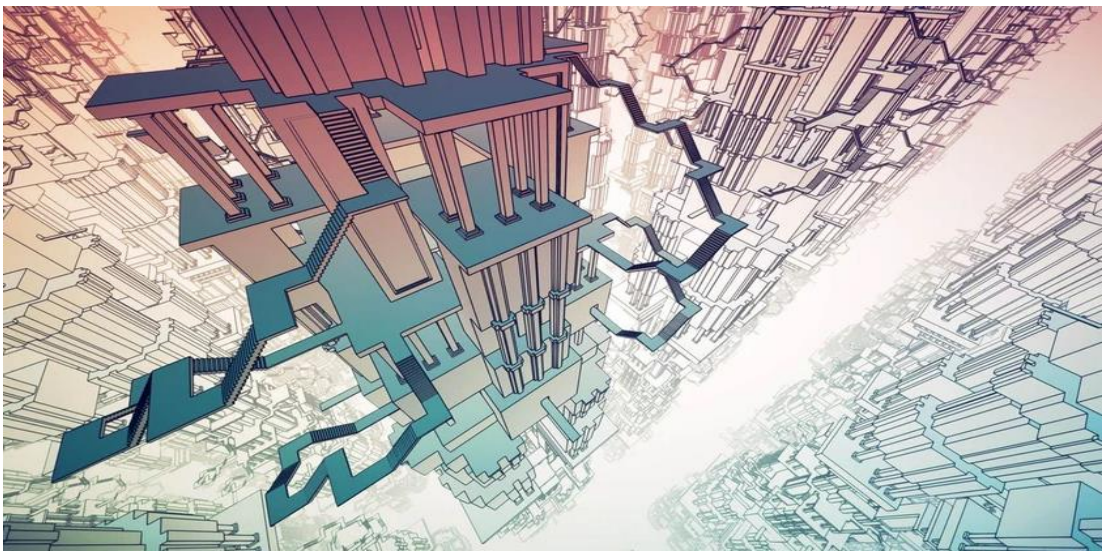

Рис. 1.8. Кадри з Manifold Garden

Досить новим баченням неевклідової геометрії, а саме геометрії Лобачевського є - HyperRouge.

Світ гри являє собою карту, поділену на полігональні сектори. Ближче до центру вони займають нормальне положення, але, наближаючись до країв карти, вони починають стискатися в нескінченність. У якому б напрямку гравець не рухався, він буде спостерігати, як простір розгортається під його ногами і згортається за ним. По суті, він нескінченний, але в той же час без проблем вписується в невелику круглу карту (Рис. 1.9).

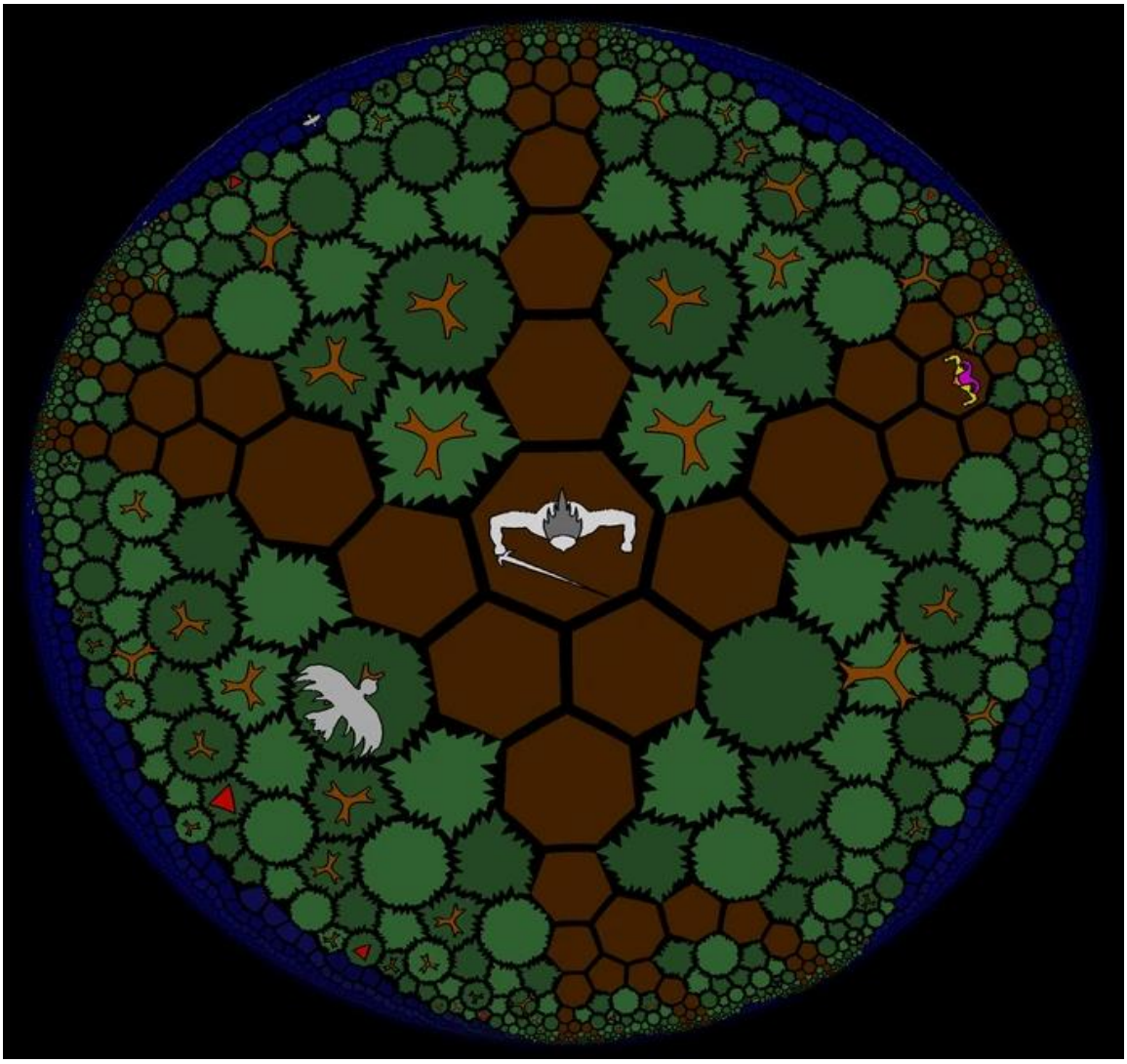

Рис. 1.9. Кадр із гри HyperRouge

Ігровий процес HyperRouge нагадує шахи - тільки ігрове поле набагато складніше. Вороги рухаються за передбачуваною схемою і намагаються поставити вам мат, а ви продумуєте свої ходи, щоб не бути загнаними в кут.

HyperRouge почався як невеликий технічний експеримент з неевклідовою геометрією, але з часом виріс у повноцінний рогалик, який отримав захоплені відгуки від більшості людей, які наважуються грати в нього. На даний момент це чи не єдиний проект, який дозволяє на практиці зрозуміти логіку неевклідового простору.

1.8.1 Проекції у неевклідовій геометрії для віртуального простору

За основу проекції у віртуальному просторі відповідають так названі портали.

Портал в науковій фантастиці і фентезі - це технологічний або магічний отвір, що з'єднує два віддалених місця розташування, розділених простором і часом.

Портал може з'єднувати місця в тій же Всесвіту (аналог телепортації); в паралельному світі (міжпросторовий портал); в минулому або майбутньому (тимчасової портал); або інші аспекти існування, такі як рай, пекло і т. д.

Опис порталів в фантастиці схоже на космологічну концепцію червоточини, і найчастіше принципи роботи порталів в творі пояснюється наявністю «кротячих нір».

Історія реальної, а не вигаданої телепортації почалася в 1993 році, коли американський фізик Чарльз Беннетт математично - за допомогою формул - довів теоретичну можливість миттєвих квантових переміщень.

Звичайно, це були суто теоретичні викладки: абстрактні рівняння, що не мають ніякого практичного застосування. Однак точно так же - математичним шляхом - вже були відкриті, наприклад, чорні діри, гравітаційні хвилі і інші явища, підтвердити існування яких експериментально вдалося набагато пізніше.

Так що розрахунки Беннетта стали справжньою сенсацією. Вчені почали активно вести дослідження в цьому напрямку - і перший успішний досвід квантової телепортації вдалося провести вже через кілька років.

Тут потрібно підкреслити, що мова йде саме про квантової телепортації, а це не зовсім те ж саме, що ми звикли бачити у фантастичних фільмах. З одного місця в інше передається не сам матеріальний об'єкт (наприклад, фотон або атом - адже все складається з атомів), а інформація про його квантовому стані. Однак в теорії цього достатньо, щоб "відновити" вихідний об'єкт в новому місці, отримавши його точну копію. Більш того, такі досліди вже теж успішно проводяться в лабораторіях - але про це трохи нижче.

У звичному нам світі цю технологію найпростіше порівняти з ксероксом або факсом: ви посилаєте не саме документ, а інформацію про нього в електронному вигляді - але в результаті у одержувача виявляється його точна копія. З тією суттєвою різницею, що у випадку з телепортацією сам відсилається матеріальний об'єкт руйнується, тобто зникає - і залишається лише копія.

Практичні досліди по телепортації почалися близько 10 років тому на Канарських островах під керівництвом австрійського фізика, професора Віденського університету Антона Цайлінгер.

У лабораторії на острові Пальма вчені створюють пару заплутаних фотонів (A і B), а потім один з них за допомогою лазерного променя відправляють в іншу лабораторію, розташовану на сусідньому острові Тенеріфе, в 144 км. При цьому обидві частки знаходяться в стані суперпозиції - тобто ми ще не "відкрили котячу коробку".

Потім до справи підключають третій фотон (C) - той, що потрібно переміщувати - і змушують його вступити у взаємодію з однією з заплутаних часток. Потім фізики вимірюють параметри цієї взаємодії (A + C) і передають отримане значення в лабораторію на Тенеріфе, де знаходиться другий заплутаний фотон (B).

Незрозуміла зв'язок між A і B дозволять перетворити B в точну копію частки C (A + C-B) - як ніби вона миттєво перемістилася з одного острова на інший, не перетинаючи океан. Тобто переміщуватися.

Антон Цайлінгер керує роботами з практичної телепортації

"Ми як би витягаємо ту інформацію, яку несе оригінал - і створюємо новий оригінал в іншому місці", - пояснює Цайлінгер, який телепорований таким чином вже тисячі і тисячі елементарних частинок.

Можливо це означає, що в майбутньому вчені зможуть таким чином телепортувати будь-які предмети і навіть людей - адже ми теж складаємося з таких частинок.

У теорії це дуже навіть можливо. Потрібно лише створити достатню кількість заплутаних пар і рознести їх у різні місця, помістивши в

"телепортаціонних кабіни" - скажімо, в Лондоні і в Москві. Ви заходите в третю кабіну, що працює як сканер: комп'ютер аналізує квантовий стан ваших частинок, порівнюючи їх із заплутаними, і посилає цю інформацію в інше місто. А там відбувається зворотний процес - і з заплутаних часток відтворюється ваша точна копія.

У реальному житті телепортація майже можлива, а у віртуальному досить реальна.

Для того щоб реалізувати телепортацію досить просто перенести ігрового персонажа у нові координати. Таким чином у момент входу у фізичну область телепорту ігровому персонажу задаються координати фізичної області другого телепорту, що і допомагає зробити ефект телепортації.

Є декілька способів реалізувати портали у Unreal Engine.

Перший спосіб оснований на зміні шейдерів, завдяки чому можна досягти ефекту порталу лише візуально. Змінивши шейдера освітленості та тіней можна досягти такого ефекту(Рис. 1.10, Рис. 1.11).

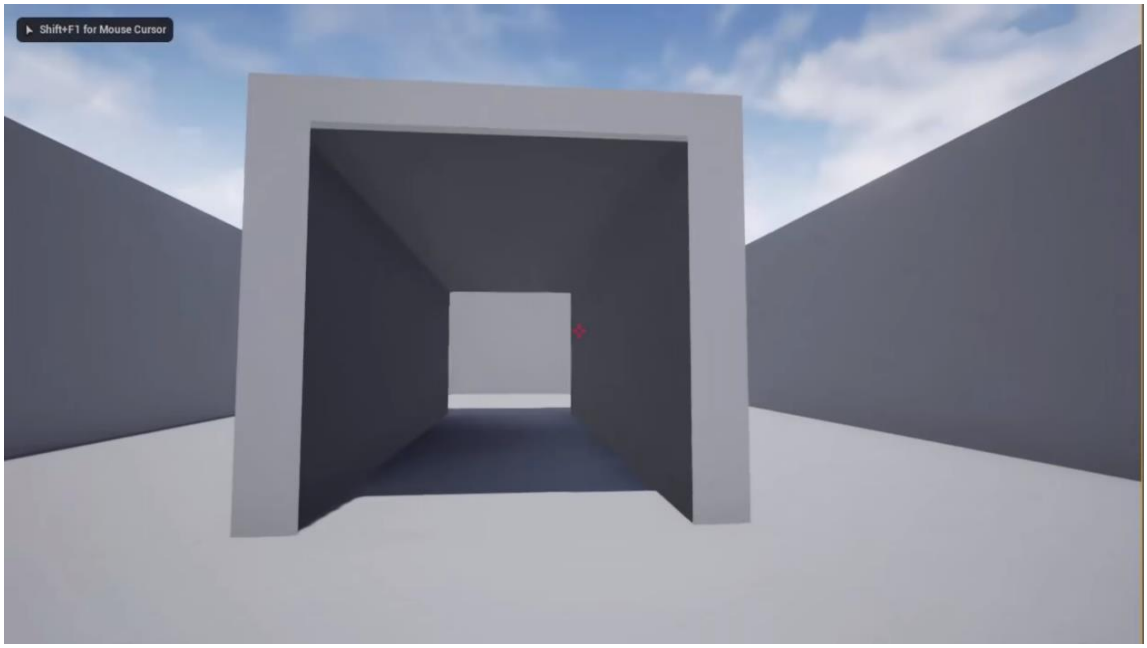

Рис. 1.10. Перший спосіб реалізації порталу спереду

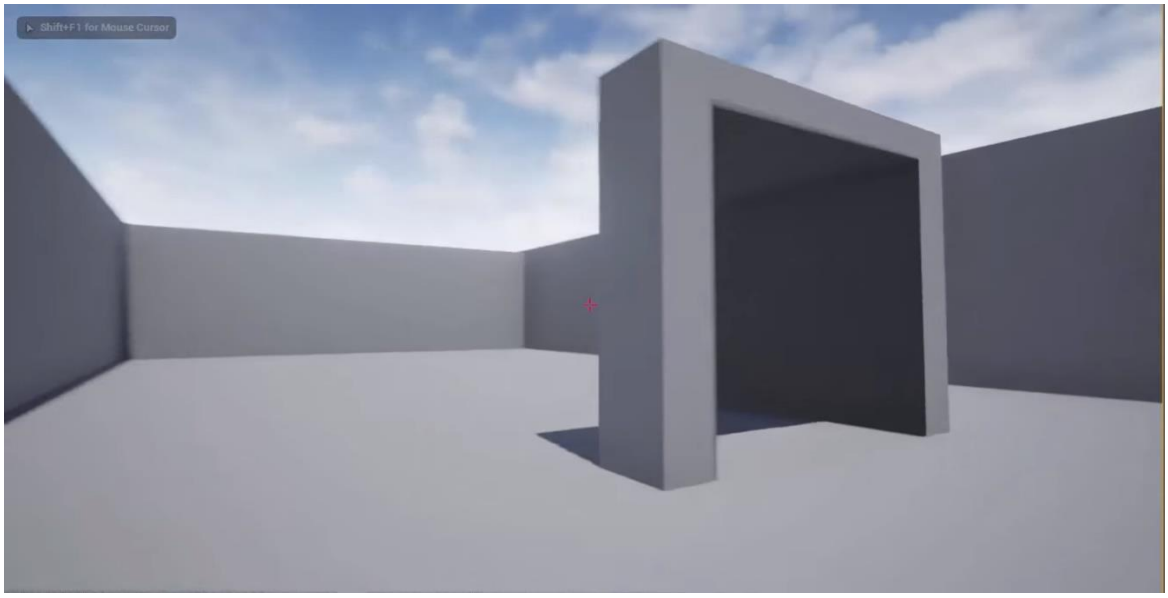

Рис. 1.11. Перший спосіб реалізації порталу збоку

Технічно змінюється лише освітлення (Рис. 1.12). Цей спосіб досить простий, завдяки ньому можна зробити нескінченні коридори та нескінчену кімнату. Для цього досить просто телепортувати ігрового персонажа у нову координату, зберігаючи поворот та висоту персонажу. Для цього способу треба телепортувати персонажа таким чином, щоб користувач бачив теж саме що бачив перед телепортацією, для того щоб не було помітно те що він змінив своє положення. Для того щоб досягти ефекту нескінченного коридору, треба просто змінити освітлення так щоб вихід з коридору був на одному й тому ж самому місці відносно камери користувача, та зменшити швидкість ігрового персонажу на стільки, щоб був ефект того, що персонаж пересувається з однією й той самою швидкістю.

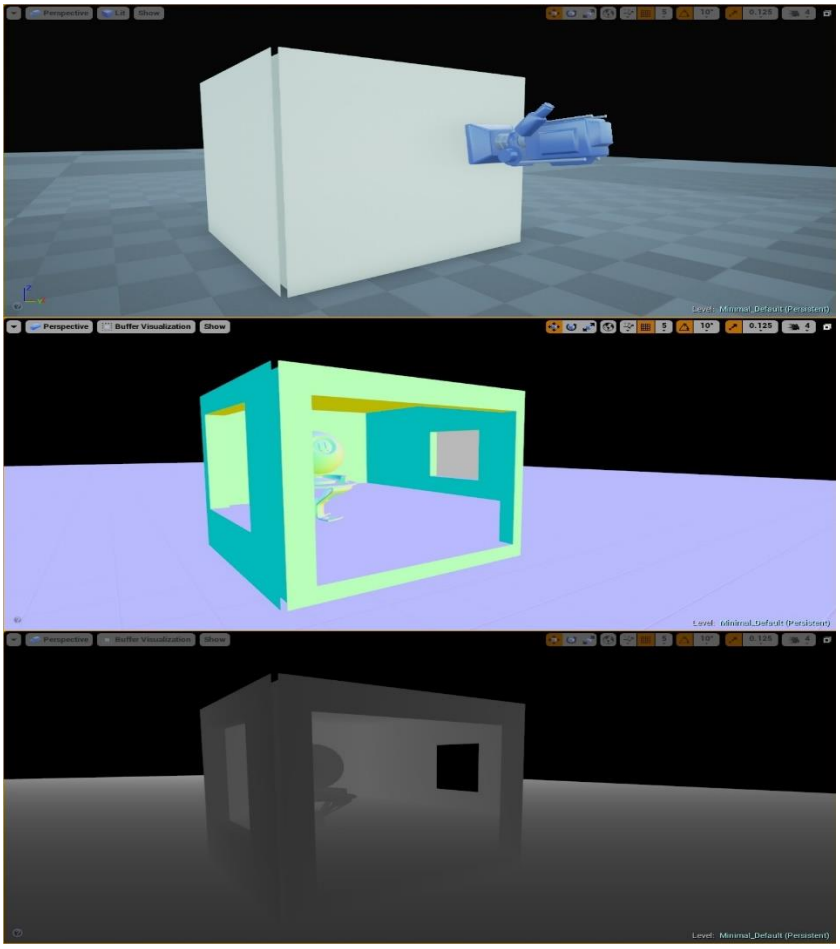

Рис. 1.12. Приклад зміни освітлення по верствам

Проблема такого підходу є у тому, що при різних рівнях висоти входу та виходу може бути неприродним положенням ігрового персонажа.

Також при зміні освітлення(динамічний перехід дня і ночі) треба проводити додаткові перерахунки цього самого освітлення, що може нашкодити оптимізації.

Беручи до уваги усі ці аспекти було прийняте рішення змінити підхід для досягання ефекту телепортації. Що значно спростило розробку, але дало додаткову навантаження на елементи відображення. Борючись з проблемою оптимізації було прийнято рішення не відображати портали до тих пір, як користувач фізично не зможе їх бачити, що трохи допомогло оптимізувати цю проблему.

1.8.2 Особливості сприйняття проекції у віртуальному просторі

Як була зазначено раніше основою для проекції у віртуальному просторі є портали.

Особливість порталів у тому, що через них можна не тільки переміщати об’єкти, а й спостерігати за тим, що відбувається по той бік порталу. Створити такий візуальний ефект не так просто. Перше, що потрібно зробити, це поставити камери на кожну з них і транслювати зображення з камери на другий портал (Рис. 1.13).

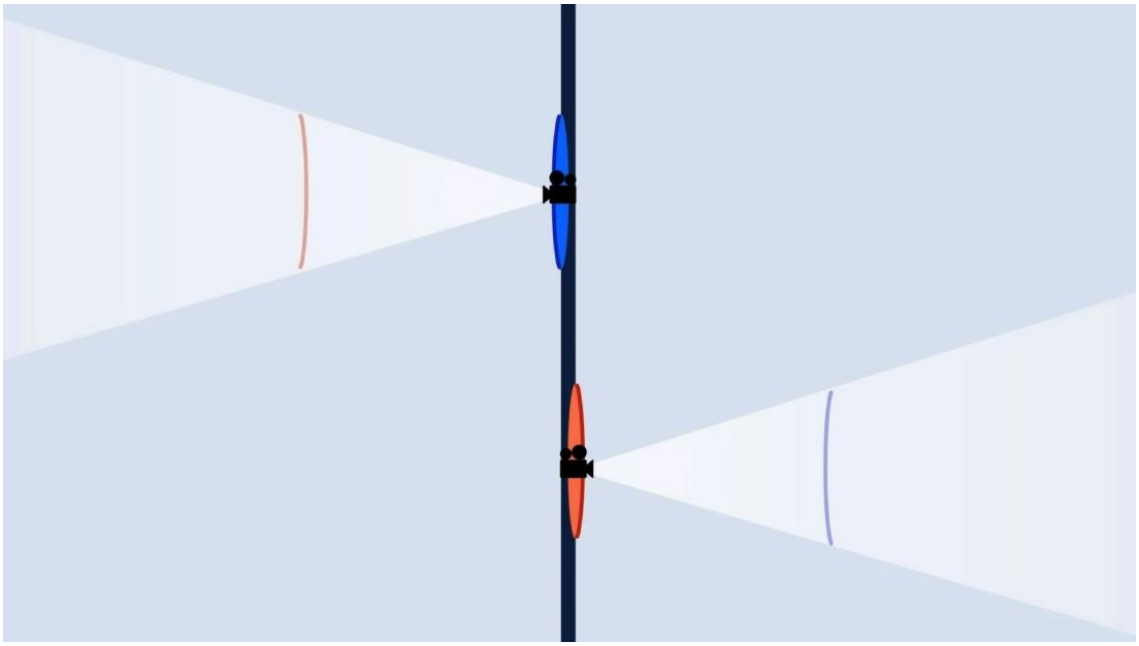

Рис. 1.13. У синій портал транслюється зображення з камери в помаранчевому порталі та навпаки

Для ефектного відображення треба змінювати зображення залежно від кута, під яким гравець дивиться на портал. Найпростіший спосіб зробити це – змінити кут камери на протилежному порталі, щоб він відповідав куту зору гравця (Рис. 1.14).

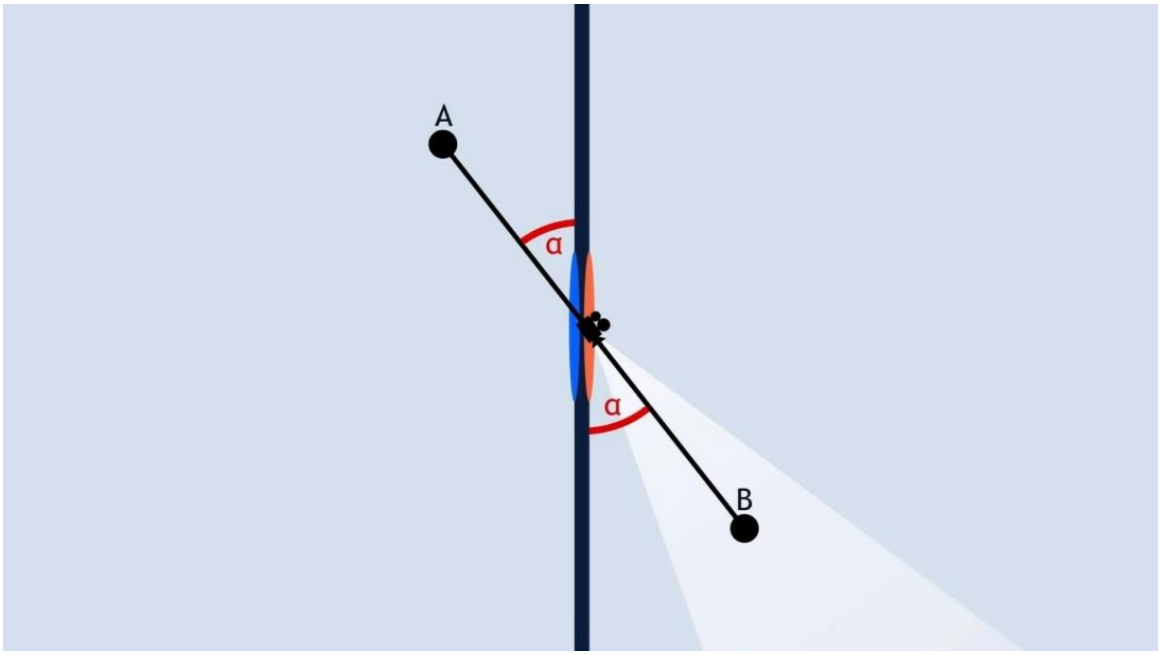

Рис. 1.14. При зміні ракурсу зображення в порталі також має змінитися

Тут можна зустріти основну проблему такого підходу, зображення у порталі не відповідає дійсності через різність поля зору камери і ігрового персонажу користувача(Рис. 1.15).

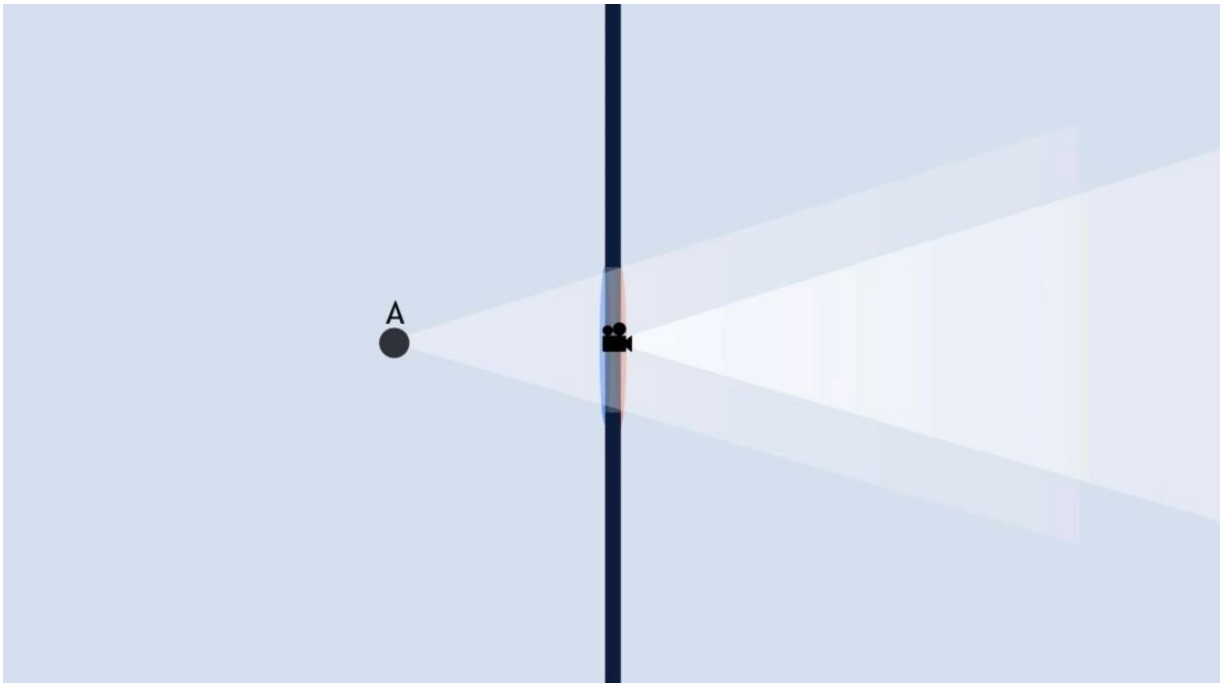

Рис. 1.15. через різність поля зору камери і ігрового персонажу користувача

Для того щоб уникнути цього ефекту, досить лише масштабувати зображення з камери у порталі на відстань ігрового персонажу користувача до самого порталу.

1.9. Застосування процедурної генерації для ігрових застосунків

Багато ігрових састосунків використовує процедурну генерацію. Це полегшує розробку сюжетного оповідання у іграх з відкритим світом та багато користувацьких ігор.

Типовими прикладами використання процедурної генерації є: створення нелюдських підземель у пригодницьких іграх (наприклад, The Legend of Zelda), отримання нового світу при кожному запуску; система, яка створює нові види зброї в шутері в космічній обстановці в залежності від дій гравця; створення повної, ігрової та збалансованої настільної гри; внутрішня процедура ігрового движка, швидко наповнюючи ігровий світ рослинами; інструмент, який дає можливість людині створювати карти для стратегічної гри, і за запитами та внесеними змінами перераховує карту для її покращення, а також пропонує варіанти, щоб зробити карту більш збалансованою та цікавою. У той же час простий редактор карт, штучний інтелект для настільної гри чи інструмент для інтеграції створеного контенту не належать до процедурної генерації.

Метою використання процедурної генерації може бути створення ігрового контенту без втручання людини (який може бути як менш дорогим, так і допомогти дизайнерам ігор у вирішенні їхніх проблем), розробка інших типів ігор (покращення показників різноманітності та відтворюваності), адаптація ігор для гравця на льоту, покращення контенту за допомогою алгоритмічних рішень, а також формалізації ігрового дизайну як широкого наукового завдання .

Перші широко відомі застосування процедурної генерації датуються початком 1980-х років, коли з обмеженими ресурсами комп’ютера можна було створювати великі й різноманітні світи – типовими прикладами є Rogue та Elite. Зовсім недавно PCG використовується в комерційних іграх: наприклад, у X-

COM: UFO Defense (1994) і Diablo (1996) PCG використовується для створення карт, генерації розташування та кількості монстрів і предметів; наріжним каменем ігрового процесу PCG є Spore, який використовує процедурну анімацію; використовується в серії Civilization для створення карт; відомими прикладами додатків є Minecraft і Spelunky (ігровий світ) і Tiny Wings (англ.) рос. (генерування текстури.

1.10 Постановка задачі

У даній кваліфікаційній роботі необхідно вирішити наступні задачі:

1. Розробити ігровий додаток на базі ігрового двигуна Unreal Engine.

2. Впровадити оптимальний алгоритм процедурної генерації до розробленого ігрового застосунку.

3. Реалізувати методи неевклідового простору до алгоритму процедурної генерації рівня додатку.

4. Проаналізувати ефективність методів процедурної генерації, зробити оцінку швидкодії алгортиму.

1.11 Висновки

Ігрові додатки на даний час можуть бути використання для навчання. Оскільки користувач може одразу використовувати теоретичні знання на практиці у віртуальному просторі, процес навчання проходить швидше.

Для покращення просторової орієнтації можна використовувати неевклідову геометрію, оскільки вона створює не типову поведінку віртуального простору для користувача. Щоб створити ігровий застосунок, достатньо використати один із існуючих ігрових двигунів, що значно спрощує розробку самого ігрового додатку. Процедурна генерація спрощує розробку неоднакових ігрових рівнів. Даний підхід надає змогу розробнику ігрового застосунку

використовувати один й той самий рівень ігрового додатку, та щоразу давати новий досвід для користувача.

Виходячи з перерахованих вище висновків, можна створити ігровий додаток, що допоможе користувачу розвинути просторову орієнтацію у віртуальному просторі та навчитись новому.

РОЗДІЛ 2
СТВОРЕННЯ ІГРОВОГО ЗОСТУСУНКУ З ВИКОРИСТАННЯМ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ТА НЕЕВКЛІДОВОЇ ГЕОМЕТРІЇ

2.1 Методи та моделі розробки ігрового застосунку на базі Unreal Engine

У ігровому двигуні реалізовано багато компонентів. Одні з таких є компоненти пересування.

Компонент Character Movement - руху персонажа

Character Movement Component дозволяє Аватар не використовувати фізику твердого тіла, щоб рухатися пішки, або бігати, стрибати, літати, падати і плавати. Він специфічний для персонажів і не може бути реалізований будь-яким іншим класом. Він автоматично додається при створення Blueprints на основі класу персонажу, але не вручну.

Властивості, які можуть бути встановлені, включають значення падаючого і ходового тертя, швидкості руху по повітрю, по воді і по землі, плавучості, гравітаційний масштаб і фізичні сили, які Персонаж може надавати на об'єкти Фізики. CharacterMovementComponent також включає в себе параметри кореневого руху, які надходять з анімації і вже трансформовані в світовий простір, готові до використання фізикою. Дивіться розділ Root Motion для отримання додаткової інформації.

Інформацію про роботу з рухом символів дивіться у розділі Setting Up Character Movement (Налаштування руху персонажа).

Компонент Projectile Movement - метального руху

Компонент Projectile Movement Component оновлює положення другого Компонента, коли він відзначається. Поведінка, таке як відскок після ударів і самонаведення до мети, підтримується цим типом Компонента. Зазвичай кореневої Компонент володіє Актора переміщається, однак може бути обраний і інший компонент (дивіться розділ SetUpdatedComponent). Якщо Updated

Component (Оновлений компонент) симулює фізику, тільки вихідні параметри запуску (коли початкова швидкість відмінна від нуля) вплинуть на метання, і фізична симуляція вступить в силу.

Приклад того, як Blueprint використовує Projectile Movement Component, показаний нижче (Рис. 2.1).

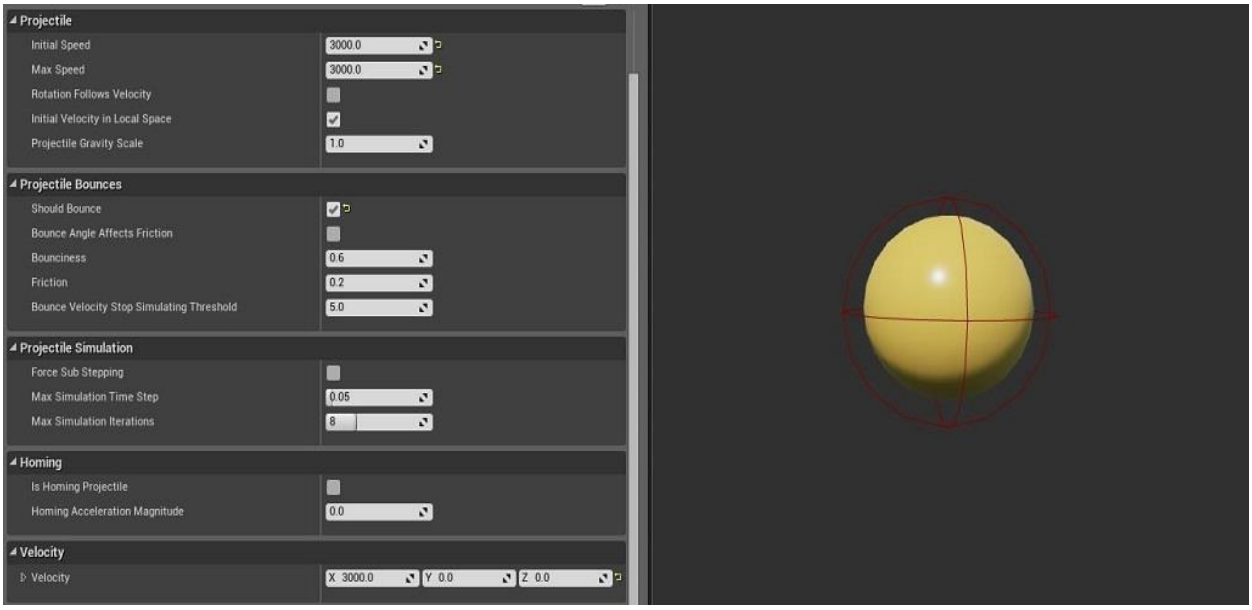

Рис. 2.1. Приклад Projectile Movement Component

Компонент Rotating Movement - обертовий рух

Rotating Movement Component виконує безперервне обертання Компоненти з певною швидкістю обертання. Обертання необов'язково може бути навіть зміщений навколо точки повороту. Важливо відзначити, що під час руху тестування зіткнення не виконується.

Приклад використання Rotating Movement Component може бути у вигляді пропелерів літака, вітряка або навіть ряду планет, що обертаються навколо Сонця.

Для пересування ігрового персонажу буда використана логіка компоненту Character Movement. Що й дало змогу зробити пересування ігрового персонажу досить гладко та плавно пересуватись по фізичному виміру, реагуючи на усі фізичні об'єкти. Такий спосіб пересування найбільш відповідний для пресування персонажу, інші ж компоненти пересування більш відповідають об'єктам

фізичного виміру з якими можлива взаємодія. Наприклад Projectile Movement більш підійде для снарядів або для імітації ігрового м'яча.

Завдяки компоненту Character Movement при натисканні на кнопку пересування (WASD) ігровий персонаж збільшує вектор направлення до тієї чи іншої сторони, що я виглядає як пересування по плоскості.

Також для реалізації відображення існують Rendering Components. Завдяки цим компонентам відбувається відображення ігрового застосунку. При зміні цих компонентів, можна стилізувати відображення. Наприклад, зробити відображення для ігрового застосунку, дії якого відбуваються у космосі. Саме за такий єфект відповідає AtmosphericFogComponent.

AtmosphericFogComponents використовуються для створення ефектів запотівання, таких як хмари або навколишній туман на рівні. Існує кілька налаштувань, які можна налаштувати для цього компонента, які можуть вплинути на те, як ефект генерується на вашому рівні при розміщенні.

Нижче наведено приклади цього типу компонента (Рис. 2.2 та Рис. 2.3), який використовується з різними значеннями для його налаштування висоти розпаду (який керує висотою розпаду щільності туману, тобто менші значення призводять до того, що туман стає щільнішим, тоді як вищі значення розріджують туман, викликаючи менший розсіювання.

Рис. 2.2 Висота розпаду щільності 0,5 (8 км)

Рис. 2.3 Висота розпаду щільності 0,35 (2,744 км)

Частіш за все використовується компонент Particle System Component.

ParticleSystemComponent дозволяє вам додати випромінювач частинок як підоб’єкт до іншого об’єкта. Додавання компонента ParticleSystemComponent можна використовувати з кількох причин: від додавання ефекту вибуху до чогось, що знищується, до додавання ефекту вогню до чогось, що ви можете підпалити. Додавши цей тип компонента до іншого об’єкта, за допомогою сценарію ви можете отримати доступ і встановити будь-який з параметрів ефекту частинок під час гри (тобто увімкнути або вимкнути ефект).

Наприклад, нижче ми маємо камеру безпеки та додали компонент ParticleSystemComponent для ефекту іскри (Рис. 2.4).

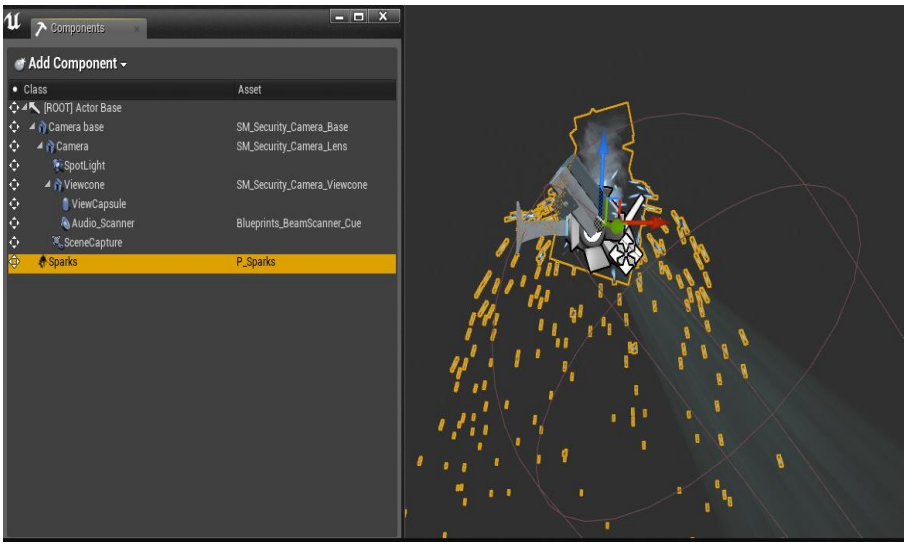

Рис. 2.4. Використання ParticleSystemComponent

Даний компонент використовується , не тільки для додання іскор. Цей компонент може бути використано для імітації дощу, диму та додавати реалістичності для руйнування.

Для імітації дощу даний компонент потрібно поєднати с іншим компонентом – VectorFieldComponent.

VectorFieldComponent використовується для посилання на векторне поле, яке є тривимірним контейнером із сіткою векторів швидкості, які можна використовувати для визначення швидкості або прискорення спрайтів GPU. Векторні поля можна використовувати для ефектів невеликого масштабу, як-от ефект часток поривів вітру до великомасштабних хуртовин на ваших рівнях (Рис. 2.5).

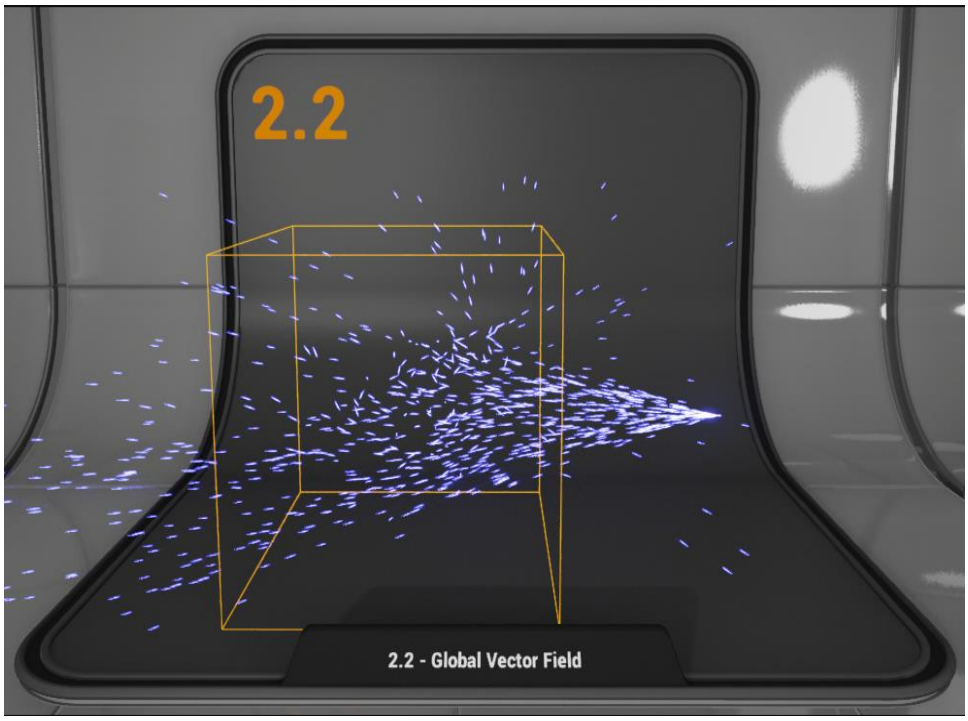

Рис. 2.5. Приклад використання VectorFieldComponent

Таким чином можна задавати та міняти напрям вітру при сильному дощу. Цей ефект є досить важливим для імітації дощу, тому що, у реальному житті вітер и злива відчуваються хаотичними. Виходячи з інформації про всі

перераховані компоненти, можна зазначити, що найкраще використання компонентів можна досягти при використанні декількох компонентів разом.

Досить важливою частиною компонентів є компоненти камери.

CameraComponent додає перспективу камери як підоб'єкт до актора. CameraComponent надасть інформацію про властивості камери, якщо ViewTarget є CameraActor або Actor, який містить CameraComponent і для параметра Find Camera Component When ViewTarget встановлено значення true.

Наприклад, ви можете використовувати CameraComponents для перемикання між кількома камерами на вашому рівні під час гри. Використовуючи Set View Target за допомогою Blend і CameraActor, ви можете перемикатися між кожною з ваших камер і використовувати властивості, визначені в CameraActor для кожної камери (включаючи поле зору, кут, ефекти постобробки тощо).

Пов'язана властивість, яка може бути встановлена для будь-якого пішака, — це взяти контроль над камерою, коли він одержимий, що призводить до того, що пішак автоматично стає цільовою об'єктом перегляду після володіння контролером гравця. Так, наприклад, якщо у вас є кілька персонажів (які є формою Pawn), між якими ви хочете перемикатися, і кожному з них призначено власний компонент CameraComponent, що забезпечує перспективу камери для перегляду, ви можете налаштувати Take Camera Control When Possessed має значення true для кожного з них і щоразу, коли ви перемикаєтеся між ними, CameraComponent для цього пішака буде використовуватися.

Прикадом такого компоненту можна взяти Spring Arm Component.

SpringArmComponent намагається підтримувати своїх дочірніх елементів на фіксованій відстані від батьківського елемента, але відтягує дочірніх елементів, якщо є зіткнення, і повертається назад, коли немає зіткнення. Як правило, SpringArmComponent використовується як «стріла камери», щоб не допустити зіткнення наступної камери гравця зі світом (без SpringArmComponent CameraComponent залишиться на фіксованій відстані, незалежно від об'єктів, які

потенційно можуть стати на шляху до його та об’єкта, до якого він прикріплений) (Рис. 2.6).

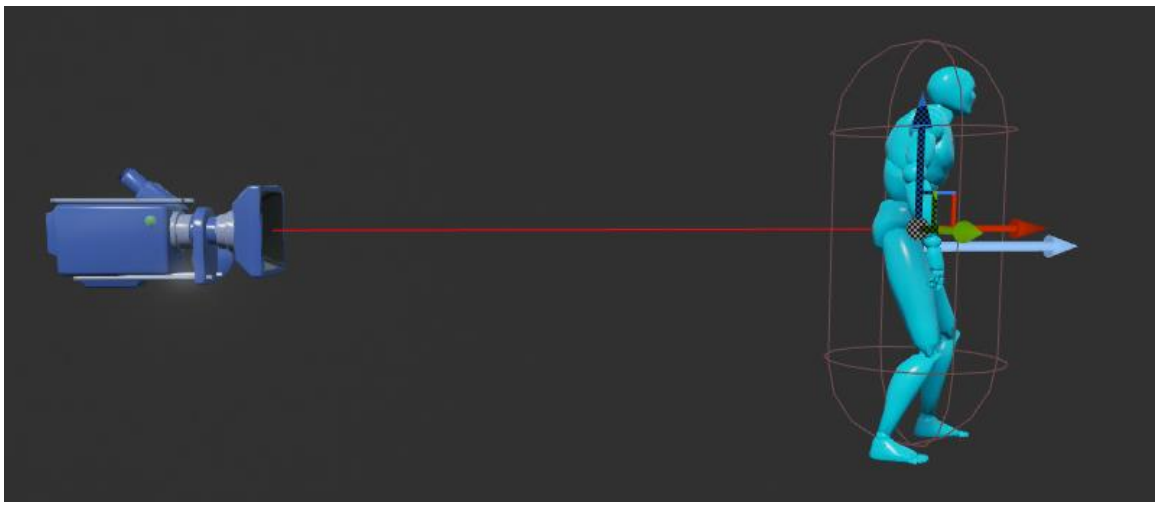

Рис. 2.6. Приклад SpringArmComponent компоненту

Мабуть найважливішим блоком компонентів можна вважати компоненти фізики.

Фізичні компоненти використовуються для впливу на будь-які об’єкти, які використовують фізику на вашому рівні різними способами.

Одним із представників фізичних компонентів є Destructible Component.

DestructibleComponent містить фізичні дані для знищуваного актора. Додаючи цей компонент як підоб’єкт, ви повинні вказати об’єкт Destructible Mesh для використання. Ви також можете замінити та вказати Ефекти Злому замість використання Ефектів Злому активу, якщо хочете.

Прикладом використання цього типу компонента може бути віконна рамка та вікно, де віконна рамка — це StaticMeshComponent, а вікно — DestructibleComponent, який гравець може вистрілити, спричинивши його розпад на частини (Рис. 2.7).

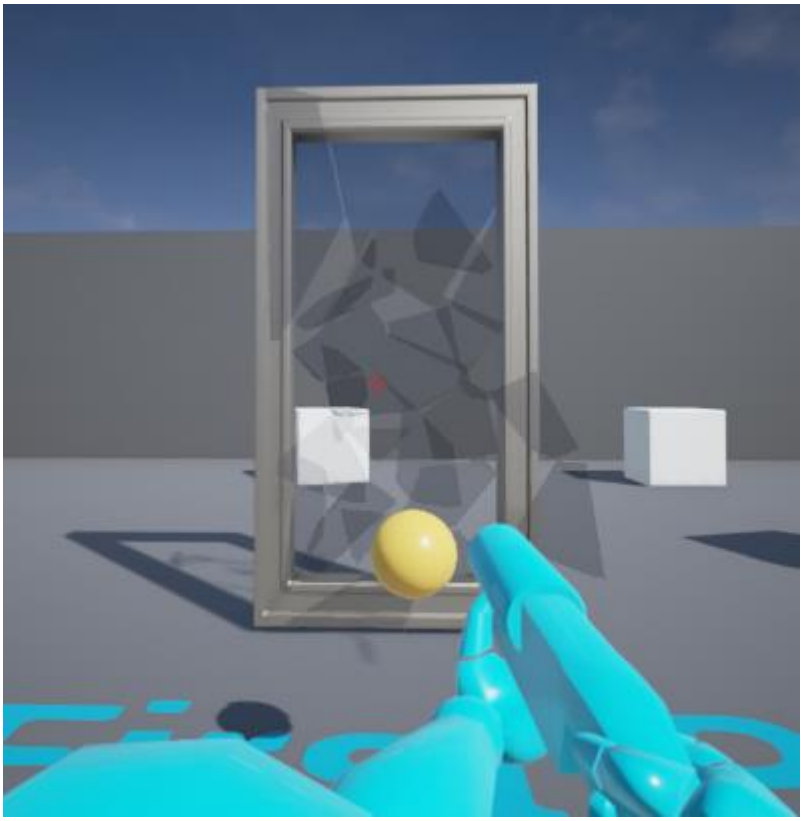

Рис. 2.7. Приклад використання DestructibleComponent

Якщо треба зв’язати фізичні властивості, слід використовувати PhysicsConstraintComponent.

PhysicsConstraintComponent – це з’єднання, яке дозволяє з’єднати два твердих тіла разом. Ви можете створювати різні типи з’єднань, використовуючи різні параметри цього компонента.

Використовуючи PhysicsConstraintComponent і два StaticMeshComponents, ви можете створювати об’єкти типу, що бовтаються, як-от гойдалку шин, важку сумку або знак, що реагує на фізику світу, дозволяючи гравцям взаємодіяти з компонентом (Рис. 2.8).

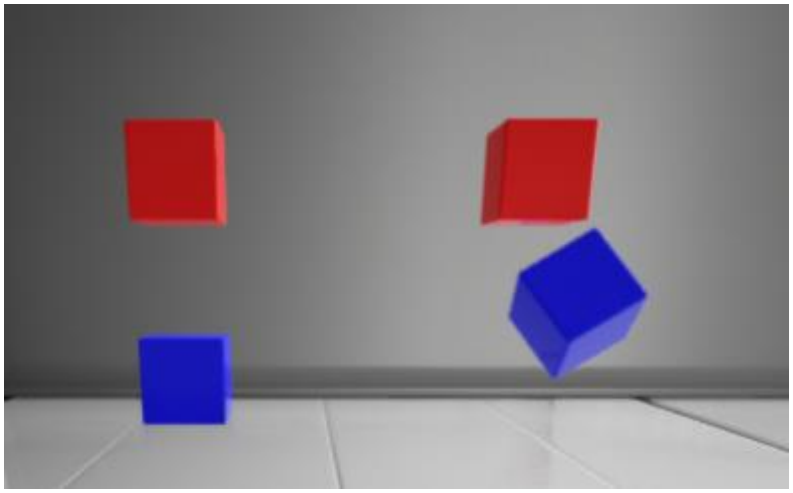

Рис. 2.8. Приклад використання PhysicsConstraintComponent

Багато ігрових додатків використовують фізику підбирання. Це також можливо і в Unreal Engine. Для цього достатньо використати Physics Handle Component.

PhysicsHandleComponent — це об'єкт для «захоплення» та переміщення фізичних об'єктів, дозволяючи об'єкту, який ви захоплюєте, продовжувати використовувати фізику. Прикладом цього може бути «гравітаційна гармата», де користувач може підбирати та кидати фізичні об’єкти (Рис. 2.9).

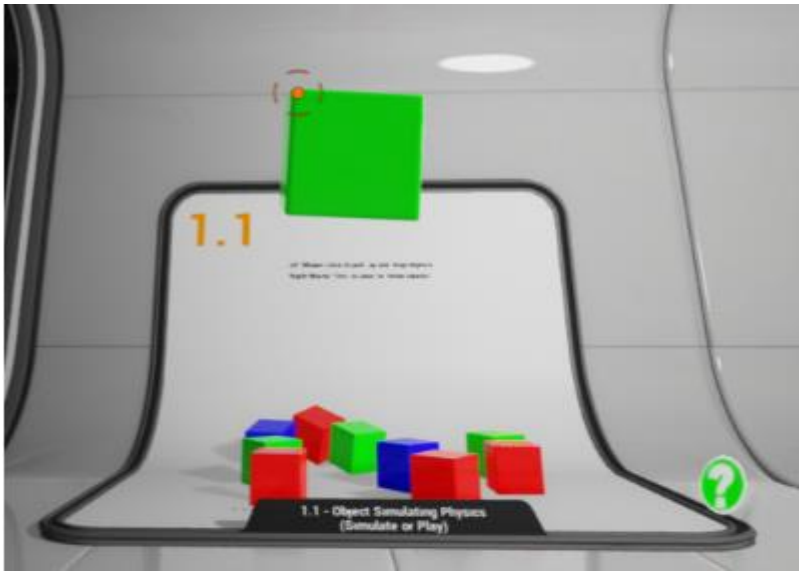

Рис. 2.9. Використання компоненту PhysicsHandleComponent

2.2 Структура та опис метрик неевклідової геометрії в ігровому застосунку

Евклідова метрика є найбільш природною функцією відстані, яка зустрічається в геометрії, що відображає інтуїтивні властивості відстані між точками. Крім того, в евклідових просторах є й інші метрики, які використовуються як в геометрії, так і в програмах. Параметрична відстань Мінковського є узагальненням деяких із цих показників; для параметра зі значенням 2 він перетворюється на евклідову метрику.

Одним із методів досягнення проекції неевклідової геометрії у ігрових застосунках є портали. Вони можуть буди лише візуальними - просто показуючи користувачу зображення по той бік, або фактично премістити користувача у ту зону відображення. Отже, як це все працює на базі ігрового двигуна Unreal Engine.

Для того щоб почати розробляти ігровий застосунок не обовязково треба знати мову програмування цього ігрового двигуна – C++. Unreal Engine використовує запатентовану технологію Blueplint. Система візуальних сценаріїв Blueprints в Unreal Engine - це повна система сценаріїв ігрових процесів, заснована на концепції використання інтерфейсу на основі вузла для створення елементів ігрового процесу в Unreal Editor. Як і у багатьох поширених мовах сценаріїв, він використовується для визначення об'єктно-орієнтованих класів або об'єктів в двигуні. Використовуючи UE4, ви часто виявляєте, що об'єкти, визначені за допомогою програми Blueprint, розмовно називаються лише "Кресленнями"(англ. Blueprint).

Отже для створення порталу можна використати цю технологію.

Для того щоб розробити портал треба створити декілька фізичних об’єктів(Actor). Данні об’єкти вже мають логіку фізики у застосунку. Щоб звязати два портали треба лище використати інформацию з камери першого порталу, та його координати положення. Таким чином можна звязати пару порталів.

На діаграмі Blueprint зображено використання данних о положені, а також логіка відображення проекції на кожному із порталів (Рис. 2.10).

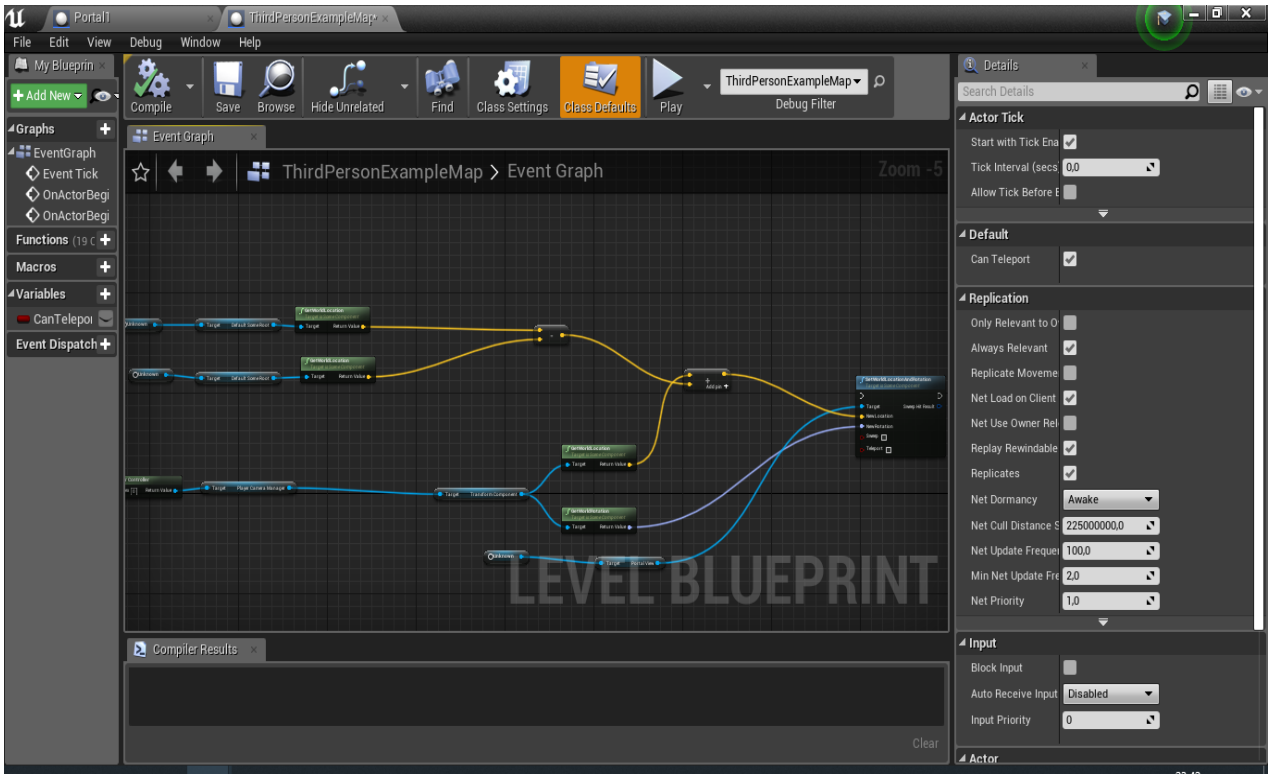

Рис. 2.10. Приклад Blueprint порталу у редакторі

Графічна частина порталу працює майже за таким же принципом(Рис. 2.11).

На графічній складовій першого порталу рендерится вся область видимості, яка знаходиться за графічною складовою другого порталу. Цю область можна виділити допоміжною віртуальною камерою. Але при такому підході є ряд проблем, які треба вирішити:

1. Не коректне зображення при повороті одного з порталів.
2. Поганий захват камери іншого порталу.
3. Не адаптивний кут відображення до напряму камери ігрового персонажу.

Для рішення усіх цих проблем потрібно було розробити математичну модель вектору направлення для порталів.

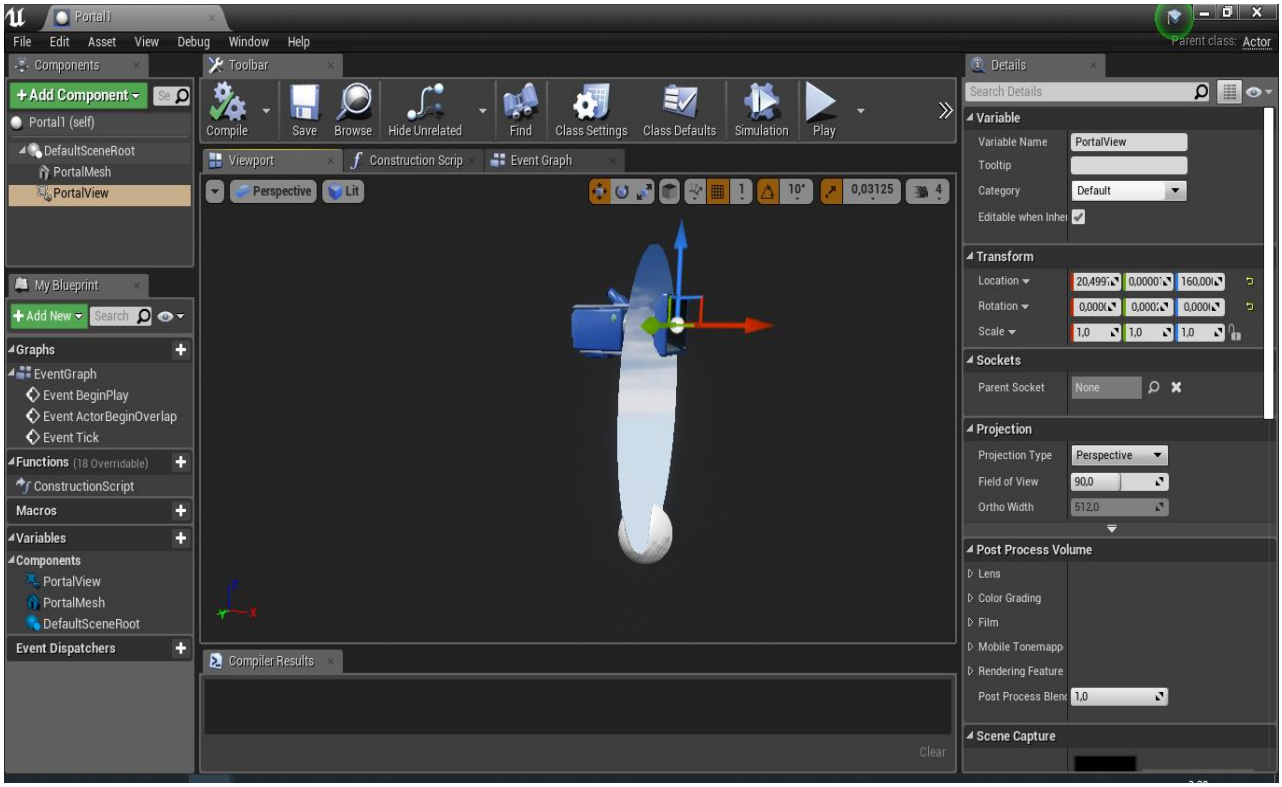

Рис. 2.11. Приклад графічної складової порталу

Таким чином перший портал отримує вектор напрямку камери ігрового персонажу та додає його до різниці векторів повороту двох порталів, цей вектор передається на камеру другого порталу, та в ній змінюється кут відображення який буде чесним відносно ігрового персонажу.

2.3 Алгоритм процедурної генерації неевклідових просторів для ігрового застосунку

Є багато різних методів для реалізації процедурної генерації у ігрових застосунках. За для того щоб обрати найкращу, треба визначитись, для чого саме потрібно використовувати процедурну генерацію. Для даного ігрового додатку, процедурна генерація підрівнів потрібна для того, щоб користувач не зміг передбачити де саме можуть з’являтися ці портали. Для того, щоб створити унікальний досвід для різних користувачів і буде використано процедурну генерацію.

Генератор процедурного рівня функціонує у відповідності з наступними фазами:

- генерація вихідної геометрії (на вибір - або «BSP», або планування приміщення);
- очищення сміттєвих розділів (таких розділів, які не можуть існувати в грі);
- будівництво зв'язків;
- очищення сміттєвих підграфів (таких груп розділів, які пов’язані між собою, але немає зв’язку з іншими розділами);
- рчищення непотрібних зв'язків (побудова основного дерева, посилання дається на мінімальне основне дерево, оскільки там є картинка, а в мінімальному генератору вона не потрібна);
- рандомізація з'єднання - відновлення деяких видалених зв'язків (для більш «людського» вигляду рівня), а також перетворення деяких інших у проходи між розділами (що «зливає» кілька секцій в одну, більш складну форму);
- генерація «сценарію» (де будуть початковий і кінцевий розділи, і який шлях потрібно пройти, щоб пройти від початкового до кінцевого).
- сптимізація з'єднань;
- створення дверей та вікон;
- виберати дію, яку потрібно виконати в цьому розділі (натисніть перемикач, підніміть ключ або знайдіть секретну стіну).

Алгоритм досить простий. Спочатку створіть прямокутник розміром із все ігрове поле.

Потім ділимо його довільно на дві частини - або по горизонталі, або по вертикалі. Ми також випадковим чином вибираємо, де буде проходити лінія поділу, і рекурсивно робимо те ж саме для нових прямокутників.

І знову і знову, до певної межі.

Потім у кожному прямокутнику виберіть «кімнату» — прямокутник такого ж розміру, як оригінал або менше.

Потім кімнати з’єднуються коридорами. Але не кожен з кожним, а трохи розумно, тому вони зберігаються в «БСП»-подібній структурі.

У результаті роботи алгоритму буде створено декілька «кімнат» та їх розташування (Рис. 2.12).

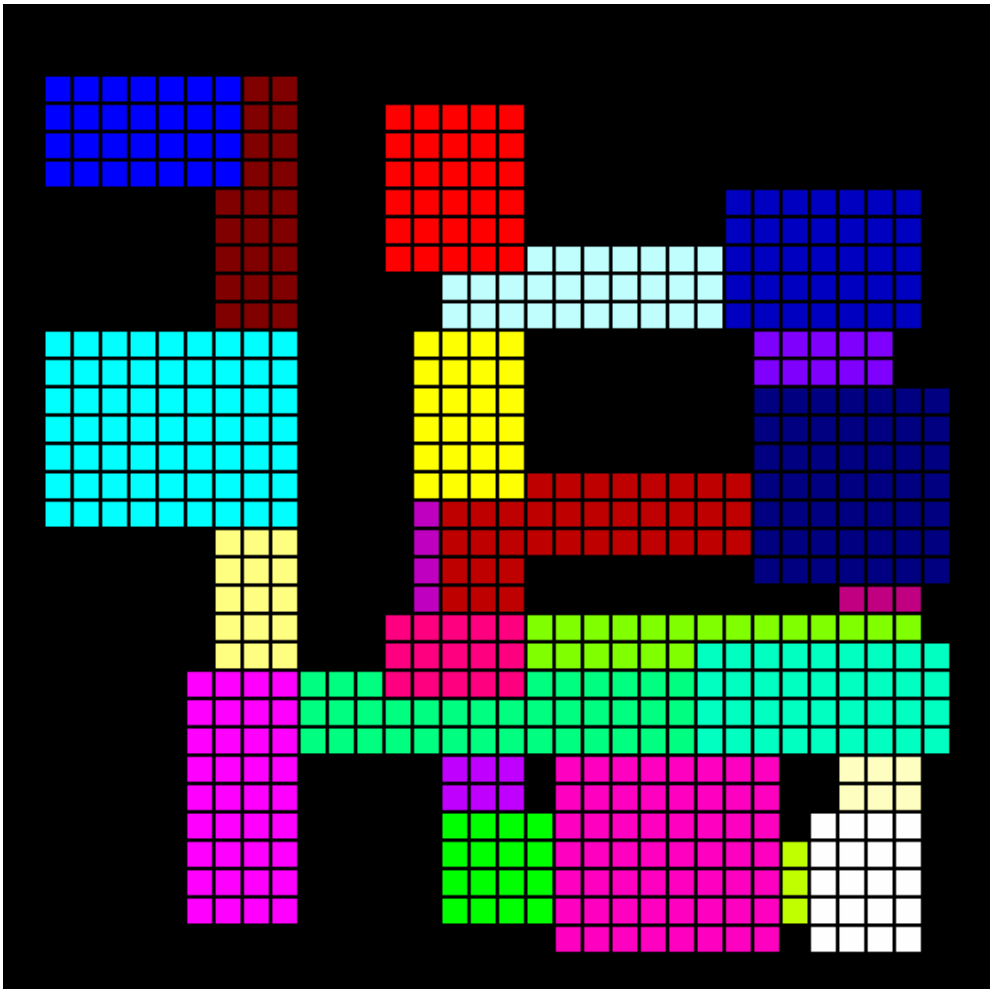

Рис. 2.12. Приклад роботи алгоритму процедурної генерації

Таким чином для того щоб кожний нових запуст ігрового додатку створював нові мапи простору достатньо лише запустити генератор процедурної генерації. Цей підхід економить багато часу на створення, розстановку, наповнення простору ігрового застосунку.

2.4 Архітектура проекції для неевклідової геометрії на базі Unreal Engine

На базі Unreal Engine використання порталів для досягнення ефекту неевклідової геометрії досить ресурсо затратне. Для того, щоб створити портал, треба зробити відображення по ту сторону порталу, яку не бачить користувач, що є причиною надлишкового навантаження графічного процесору. Саме через цю особливість слід використовувати самі портали з обережністю. В іншому випадку, система просто не зможе реалізувати потенціал ігрового застосунку.

Для того щоб проекція була коректною, треба використати декілька вимог:

- правильно розташувати портали відносно один одного;
- проводити векторні перетворення з інверсією;
- дотримуватись чіткої моделі подій у Unreal Engine;

Правильно розташувати портали відносно один одного. Для того, щоб портали не відображали картинку з самих себе, треба розташувати їх в одному напрямку. Якщо портали будуть розташовані камера проекції один до одного, вони ввійдуть у рекурсію і будуть відображати проекцію себе. Такий варіант не досить практичний (Рис. 2.13).

Рис. 2.13. Приклад неправильного розташування порталів

Проводити векторні перетворення з інверсією. Для того щоб коректо відображати те що потрібно портали використовують вектори проекції та вектор ігрового персонажу користувача. Отже, щоб уникнути проблеми з проекцією на два портали одного й того зображення слід робити інверсію для другої проекції.

Дотримання чіткої моделі подій у Unreal Engine. Як і багато інших двигунів в Unreal Engine використовується модель подій. Вона розроблена для того, щоб було досить зручно використовувати цей ігровий двигун.

Події — це вузли, які викликаються з коду гри, щоб почати виконання окремої мережі в EventGraph. Вони дозволяють Blueprints виконувати серію дій у відповідь на певні події, що відбуваються в грі, наприклад, коли гра починається, коли скидається рівень або коли гравець отримує пошкодження.

До подій можна отримати доступ у Blueprints, щоб реалізувати нові функції або замінити чи розширити функціональність за замовчуванням. В рамках одного EventGraph можна використовувати будь-яку кількість подій; хоча може використовуватися лише один з кожного типу (Рис. 2.14).

Подія може виконувати лише один об’єкт. Якщо треба ініціювати кілька дій з однієї події, потрібно буде об’єднати їх лінійно.

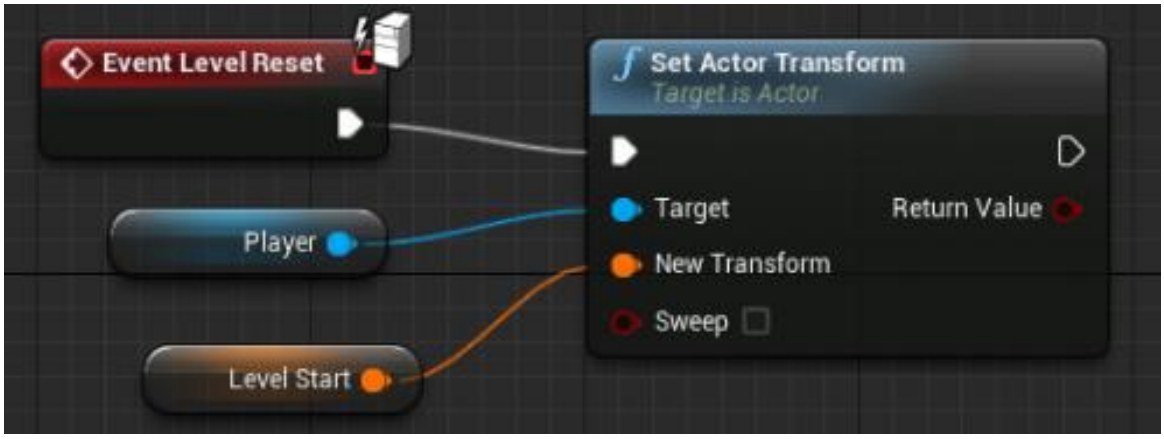

Рис.3.14. Приклад використанння подій у Unreal Engine

Якщо усі ці вимоги були виконані, то проекція порталів буде працювати правильно (Рис. 2.15).

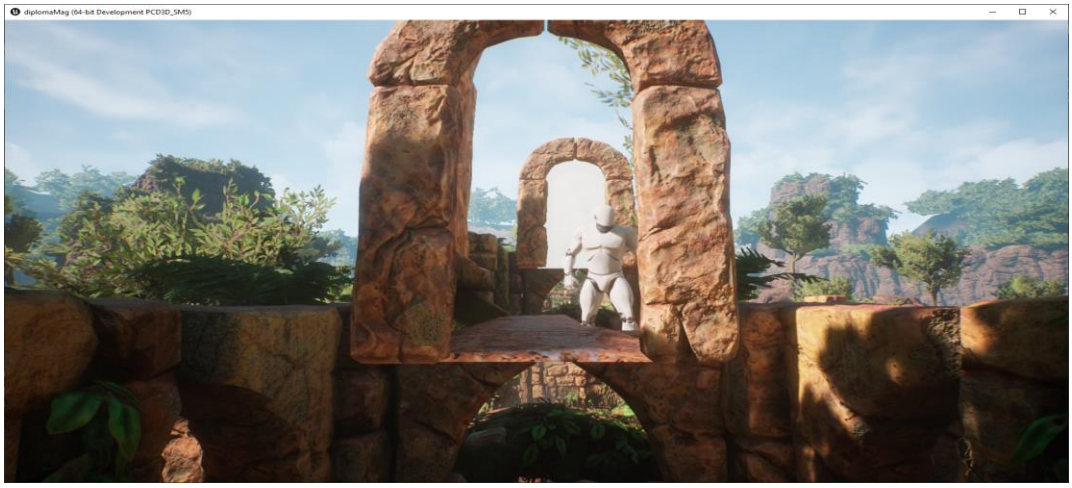

Рис.3.15. Приклад правильної роботи порталі

Отже як це все працює. Треба взяти позиції обох порталів, перетворити їх у вектори. Після цього потрібно взяти положення ігрового персонажу користувача і також перетворити на вектор. Далі різницю векторів положення порталів треба додати до вектору положення ігрового персонажу. В завершення потрібно лише передати цю інформацію до екрану відображення порталу. Для даного блоку був розроблений Blueprint, що візуально показує усі взаємодії (Рис. 2.16).

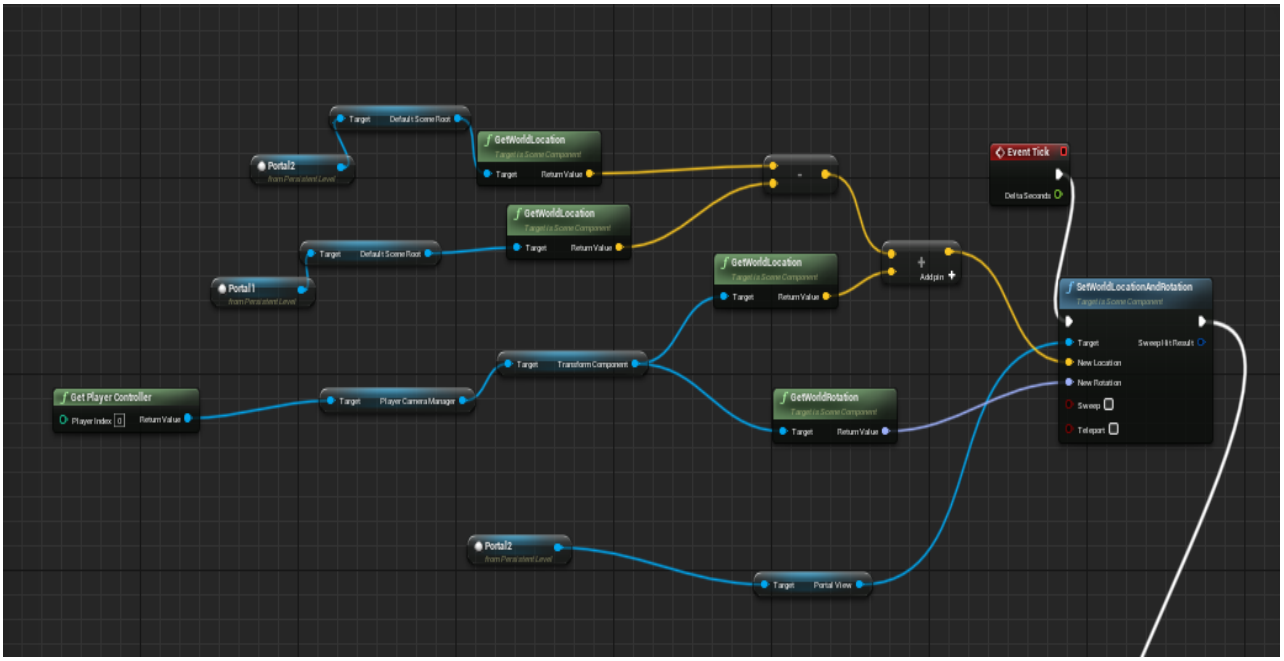

Рис. 2.16. Приклад роботи порталу у Blueprint

При правильних розташуваннях і векторних перетвореннях усіх позицій портали будуть працювати досить швидко не завантв систему користувача.

2.5 Висновки

Unreal Engine є досить потужним ігровим двигуном, на сьогоднішній час. На базі нього можна створювати ігрові додатки різних жанрів, графічних особливостей та зробити підтримку для усіх існуючих ігрових платформ.

Для отримання ефекту неевклідової моделі для віртуального простору, було використано портали. Данна технологія працює зображення з однієї частини ігрового простору у другу, таким чином можна впливати на відчуття простору користувача.

Процедурна генерація була використана для створення неоднакового досвіду як для одного користувача, так і для декількох.

Створення одного ігрового додатку з використанням вищезгаданих технологій може надати користувачу незабутній досвід, та розвити його когнітивні навички у просторовому орієнтуванні.

РОЗДІЛ 3
ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ МЕТОДІВ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ НЕЕВКЛІДОВИХ ПРОСТОРІВ В ІГРОВОМУ ДОДАТКУ

3.1 Критерії оцінювання ефективності ігрових застосунків

Для оцінки ефективності ігрових застосунків існує багато критерій. За звичай оцінка є суб'єктивною, тобто кожен із користувачів сам оцінює свій досвід використання конкретного ігрового застосунку.

Серед постійно перерахованих критеріїв можна виділити швидкість роботи ігрового застосунку та якість комп'ютерної графіки.

1. Швидкість роботи ігрового застосунку зазвичай оцінюється кількістю кадрів у секунду.

Кількість кадрів у секунду показує, скільки кадрів (окремих зображень) показує ваш монітор або телевізор кожну секунду. Чим вища частота кадрів, тим плавнішою та чуйнішою стає гра. З іншого боку, гра стає менш приємною при низькому FPS. Можливо, вам іноді друзі скаржилися, що не можуть грати через «слайд-шоу на екрані». Це означає, що вони мають низьке значенна кількісті кадрів у секунду.

У відеоіграх частота кадрів залежить як від розробника, так і від гравця. Творець гри повинен переконатися, що продукт випущено без технічних проблем, а покупець повинен бути впевнений, що у нього є платформа, яка дозволить грі працювати з прийнятною частотою кадрів (на даний момент це 30 FPS, але ситуація така швидко змінюється). Одним із найяскравіших прикладів 2021 року є Cyberpunk 2077. Довгоочікувана гра мала технічні проблеми на консолях попереднього покоління (PlayStation 4 і Xbox One) і не могла забезпечити геймеру стабільні 30 кадрів в секунду.

При цьому гравець повинен бути впевнений, що його персональний комп'ютер налаштований на запуск певної гри. Отже, на частоту кадрів на ПК впливають:

- відео Карта;
- ЦП;
- ОЗП.

Найчастіше проблеми FPS виникають через нездатність відеокарти відображати велику кількість окремих кадрів. Через це частота кадрів стає низькою, і гравець відчуває проблеми. Процесор і оперативна пам’ять мають менший вплив на FPS, але без них ви не можете досягти стабільності гри. Таким чином можна оцінити розроблений ігровий застосунок (Рис. 3.1).

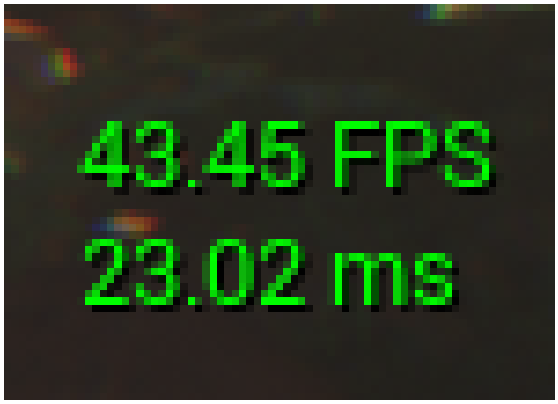

Рис. 3.1 Зображення FPS у розробленому ігровому додатку

Максимальне значення FPS для ігрового додатку на базі Unreal Engine – 60. Отже, розроблений ігровий додаток має досить великий показник FPS.

2. Якість комп’ютерної графіки.

Оскільки ігровий додаток розроблений у Unreal Engine Editor, що є еталоном для розробки високоякісного ігрового додатку (AAA). У ігровому додатку використовується технології покращення графіки:

- трасування променів світла;
- багатоядерна пост обробка;
- система покращеного згладжування текстур;

На наведених нижче рисунках 3.3 та 3.4 можна самостійно оцінити рівень комп’ютерної графіки у ігровому застосунку.

Рис. 3.3 Приклад якості графики ігрового додатку на базі Unreal Engine

Рис. 3.4 Приклад якості графики ігрового додатку на базі Unreal Engine

3.2 Ефективність запропонованого методу процедурної генерації

Будь-який процедурний генератор має два основні ступені свободи. По-перше, це складність алгоритму генерації. Це можна назвати «свободою в глибині». По-друге, різноманітність результатів. Або «свобода в ширину».

Генератор розширюється в глибину, змінюючи свій код. Це неминуче призводить до повної зміни результату генерації, навіть з однаковою зернистістю у версіях до та після ускладнення.

Оскільки усі алгоритми мають таку оцінку, як складність за часом, можна оцінити використаний алгоритм процедурної генерації.

В ігровому застосунку було використано алгоритм процедурної генерації – BSP-дерево. Даний алгоритм використовує структуру задачі комівояжера а, складність якого дорівнює $2^{O(n)}$.

Таким чином алгоритм процедурної генерації неевклідових просторів можна вважати досить швидким. Застосовуючи рисунок 3.4, можна наочно оцінити шкидкість роботи алгоритму.

Рис. 3.3 Результат роботи алгоритму процедурної генерації

3.3 Висновки третього розділу

Оскільки основна мета проекту – це створення ігрового застосунку, що зможе покращити просторове орієнтування користувача у віртуальному просторі. Для того, щоб користувач не зміг застосовувати здобутий досвід попередніх рівнів, основною ідеєю стало розробка унікальних ігрових блоків, які не схожі між собою. Таким чином, можна досягти ефекту покращення просторового орієнтування користувача, в унікальних умовах.

Отже використання процедурної генерації для ідеї покращення просторового орієнтування користувача є найкращим із методів навчання через гру.

РОЗДІЛ 4
ЕКОНОМІКА

4.1. Визначення трудомісткості розробки програмного забезпечення

Початкові дані:

1. Передбачуване число операторів програми – 1048.
2. Коефіцієнт складності програми – 1,3.
3. Коефіцієнт корекції програми в ході її розробки – 0,07.
4. Годинна заробітна плата програміста– 58 грн/год.
5. Коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2.
6. Коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,2.
7. Вартість машино-години ЕОМ –13 грн/год.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{ , людино-годин,} \qquad (4.1)$$

де t_o- витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a- витрати праці на розробку блок-схеми алгоритму;

t_n-витрати праці на програмування по готовій блок-схемі;

$t_{отл}$-витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмногму забезпеченні, яке розробляється.

Умовне число операторів (тегів):

$$Q = q \cdot C \cdot (1 + p), \tag{4.2}$$

де q - передбачуване число операторів (1300);

C - коефіцієнт складності програми (1,6);

p - коефіцієнт корекції програми в ході її розробки (0,05).

Звідси умовне число операторів в програмі:

$$Q = 1{,}3 \cdot 1048 \cdot (1 + 0{,}05) = 1430{,}52$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k}\ \text{, людино-годин,}$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності.При стажі роботи від 3 до 5 років він складає 1,2.

Приймемо збільшення витрат праці внаслідок недостатнього опису завдання не більше 50% (B = 1,2). З урахуванням коефіцієнта кваліфікації k = 1,2, отримуємо витрати праці на вивчення опису завдання:

$$t_u = \frac{(1430 \cdot 1{,}2)}{(75 \cdot 1{,}2)} = 19 \text{ людино-годин}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20\ldots25)\cdot k}\text{ , людино-годин,} \qquad (4.3)$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (4.3), людино-годин:

$$t_a = \frac{1165}{(20\cdot 1{,}2)} = 49 \text{ людино-годин.}$$

Витрати на складання програми по готовій схемі:

$$t_n = \frac{Q}{(20\ldots25)\cdot k}\text{ , людино-годин.}$$

$$t_n = \frac{(1165\cdot 1{,}2)}{(20\cdot 1{,}2)} = 58 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5)\cdot k}\text{ , людино-годин.}$$

$$t_{отл} = \frac{1165}{(5\cdot 1{,}2)} = 194 \text{ людино-годин.}$$

- за умови комплексного налагодження завдання:

$$t^{k}_{отл} = 1{,}5 \cdot t_{отл} \text{ , людино-годин.}$$

$$t^{k}_{отл} = 1{,}5 \cdot 194 = 291 \text{ людино-годин.}$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_{д} = t_{др} + t_{до} \text{ , людино-годин,}$$

де $t_{др}$-трудомісткість підготовки матеріалів і рукопису:

$$t_{др} = \frac{Q}{(15..20) \cdot k} \text{ , людино-годин,}$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0{,}75 \cdot t_{др} \text{ , людино-годин.}$$

Підставляючи відповідні значення, отримаємо:

$$t_{др} = \frac{1165}{(18 \cdot 1{,}2)} = 54 \text{ людино-годин.}$$

$$t_{до} = 0{,}75 \cdot 316 = 41 \text{ людино-годин.}$$

$$t_{д} = 54 + 41 = 95 \text{ людино-годин.}$$

Повертаючись до формули (4.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

$$t = 50 + 22 + 49 + 58 + 194 + 95 = 468 \text{ людино-годин.}$$

4.2. Витрати на створення програмного забезпечення

Витрати на створення програмного забезпечення $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{ПО} = З_{ЗП} + З_{МВ} \text{ , грн.}$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР} \text{ , грн,}$$

де: t - загальна трудомісткість, людино-годин;

$С_{ПР}$ - середня годинна заробітна плата програміста, грн/година

З урахуванням того, що середня годинна зарплата програміста становить 58 грн / год, отримуємо:

$$З_{ЗП} = 468 \cdot 58 = 27{,}144 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{мв} = t_{отл} \cdot С_{мч} \text{ , грн,} \qquad (4.4)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год;

$С_{мч}$ - вартість машино-години ЕОМ, грн/год (13 грн/год).

Підставивши в формулу (4.4) відповідні значення, визначимо вартість необхідного для налагодження машинного часу:

$$З_{мв} = 224 \cdot 13 = 2912 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 32{,}760 + 2{,}912 = 35{,}672 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot D_p} \text{міс.}$$

де B_k- число виконавців (дорівнює 1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні F_p=176 годин).

Звідси витрати на створення програмного продукту:

$$\text{Т} = \frac{468}{(1 \cdot 176)} \approx 3 \text{ міс.}$$

4.3 Маркетингові дослідження ринку

Ігри дають можливість отримати досвід, який нам недоступний у реальному житті. Це стосується не тільки битв з демонами, дослідження фантастичних світів і космічних польотів. Деякі ігри дають можливість переосмислити логіку навколишнього світу та відчути дію нових фізичних законів.

Спотворення перспективи, просторові аномалії та подорожі через портали – усі ці ігрові прийоми традиційно об’єднуються в концепцію неевклідової геометрії. У цій статті ми розберемося, як вони працюють.

Використання неевклідової геометрії у ігрових додатках досить обмежено. Це зумовлено тим, що данна технологія досить ресумістка. Але данна технологія може допомогти користучам розвити когнітивні навички.

Ідеї неможливої архітектури, спотвореного простору та нетипової навігації — чудові ідеї для ігор-головоломок. Це жанр, у якому неевклідова геометрія показала себе у всій красі.

Основне завдання Unreal Engine — спростити створення якісного проекту, в тому числі з ігровим і стабільним мультиплеером. Велика кількість активів і можливість керувати не тільки механікою, а й графікою гри є однією з головних особливостей UE.

Інші характеристики, які відрізняють цей двигун від інших, включають:

1. Повний готовий набір інструментів. Потрібно лише встановити середовище розробки та запустити його – усі необхідні функції вже будуть в Unreal Engine.

2. Розробка на C++. Ця мова програмування хоч і складніша для вивчення, ніж той же C# або Python, але працює набагато швидше. Це покращує якість та продуктивність проекту в остаточній версії.

3. Візуальний сценарій. Система Visual Scripting Blueprints дозволяє створювати ігри навіть для тих, хто не знає мови C++. І хоча для кращого результату код все одно доведеться редагувати самостійно, швидкість створення базових об’єктів можна значно збільшити.

Таким чином, постійне вдосконалення Unreal Engine дозволяє розробнику отримати найвищий рівень якості мобільних продуктів з мінімальними вкладеннями часу та зусиль.

4.4 Висновки

Ігровий додаток розроблений з метою покращення кругозору користувача, та скрасити його дозвілля. Вартість даного додатку становить 35,672 грн. та не потребує додаткових витрат при розробці проекту. Очікуваний

час розробки - 3 місяці. Цей термін пов'язаний зі значною кількістю операторів і включає в себе час для дослідження та розробки алгоритму розв'язання задачі, розробку дизайну, створення веб-сайту та підготовку документації. Оскілки ігрові додатки, які поєднують у собі гру та покращення кругозору є достатньою рідкістю, даний ігровий застосунок має досить позитивну оцінку маркетингового дослідження. Як і ігрові додатки, що використовують не типові фізичні явища, даний ігровий додаток, створює унікальний віртуальний простір для кожного користувача.

ВИСНОВКИ

В результаті виконнаня кваліфікаційної роботи було розроблено ігровий додаток, що використовує методи процедурної генерації неевклідового простору. Розроблений ігровий додаток створює проекцію віртуального оточення для користувача, базуючись на послідовному використанні основ неевклідової геометрії та допомагає розвитку когнітивного сприяття користувача через гру. Ралізована оптимізація ресурсів ігровго застосунку, та оптимізація алгоритмів процедурної генерації, завдяки чому

Реалізований проект призначений для розважальних цілей але допомагає користувачу когнітивні навички для орієнтування у просторі. Програма працює під керування ОС Linux, яка широко використовується цільовою аудиторією продукту. Розробка велась на високорівневій мові програмування C++, яка надає змогу досить точно оперувати ресурсами й дозволила досягти достатніх результатів в швидкості роботи програми.

Для аналізу ефективності методів прицедурної генерації було розглянуто ,впроваджено та протестовано декілька алгоритмів. Серед всіх алгоритмів був обрано, та оптимізовано найшвидший.

В розділі «Економіка» визначено трудомісткість розробки програмного забезпечення (468 чол-год), підраховані витрати на створення програмного забезпечення (35,672 грн.) і гаданий період розробки (5 міс.).

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ебботт Е. Флатландія. Бюргер, Сферландія. / Е. Ебботт - "Амфора", 2015 – с.140-194,

2. Каган В. Ф. Основания геометрии. ГІТТЛ (частина 1) / В. Ф. Каган - 1949 – с.368-394.

3. Гиндикин С. Г. Рассказы о физиках и математиках. / С. Г. Гиндикин - МЦНМО, 2006 – с. 365-382.

4. Лобачевский Н. И. Геометрические исследования по теории параллельных линий. / Н. И. Лобачевский, Академия Наук СССР, 1945 – с. 32-76.

5. Об основах геометрии. Сборник классических работ по геометрии Лобачевского развития и ее идей. // М.: Гостехиздат, Государственное издательство, 1956. - с. 208-487.

6. Винберг Э. Б. Дискретные группы движений пространств постоянной кривизны / Э. Б. Винберг, О. В. Шварцман // Геометрия – 2, Итоги науки и техн. Сер. Соврем. пробл. мат. Фундам. Направления. - М.: ВИНИТИ, 1988. – с. 137–186

7. Саттер Г. Новые сложные задачи на C++. Серия "C++ In-Depth" / Г. Саттер - Вильямс, 2015. - с.204–272.

8. Офіційна документація Unreal Engine C++. URL: https://docs.unrealengine.com/en-US/API/index.html (дата звернення 4.10.2021)

9. Офіційна документація Unreal Engine Blueprints

URL: https://docs.unrealengine.com/en-US/Engine/Blueprints/index.html (дата звернення 4.10.2021)

10. Будай А. Дизайн-патерни — просто, як двері - онлайн підручник. URL: https://sites.google.com/site/designpatternseasy/ (дата звернення 7.10.2021)

11. Скотт М. Эффективное использование C++. 55 верных способов улучшить структуру и код ваших программ - ДМК-Пресс, 2017. - с. 284–300.

12. Гамма Э. Рефакторинг. Улучшение существующего кода / Э. Гамма, М. Фаулер, К. Бек, Дж. Брант, У. Апдайк, Д. Робертс - Символ-Плюс, 2008. - с.387–432.

13. Александреску А. Современное проектирование на C++ / А. Александреску - Диалектика-Вильямс, 2019. - с.312– 336.

14. Шилдт Г. Java. Полное руководство. 8-е изд. — М.: Вильямс, 2012. — 1104 с.

15. Freeman B. NET 4.5 Parallel Extensions Cookbook / B. Freeman. – Published by Packt Publishing Ltd., UK. – 320 p.

16. Куксон А. Разработка игр на Unreal Engine 4 за 24 часа / А. Куксон, Р. Даулингсока, К. Крамплер. - Бомбора, 2019. - с. 243-279.

17. Shannon T. Unreal Engine 4 for Design Visualization / T. Shannon. - Addison-Wesley Professional, 2017. - pp.185-195.

18. Nixon D. Beginning Unreal Game Development / D. Nixon. - Apress, 2020. - pp. 315-348.

19. Cordone R. Unreal Engine 4 Game Development Quick Start Guide / R. Cordone. - Packt, 2019. - pp.120-170.

20. Шелл Дж. Геймдизайн. Как создать игру, в которую будут играть все / Дж. Шелл. - Альпіна Паблішер, 2019. - с.284-375.

21. Джейсон Ш. Кровь, пот и пиксели. Обратная сторона индустрии видеоигр / Ш. Джейсон. - Форс Україна, 2020. - с. 50-160.

22. Костер Р. Разработка игр и теория развлечений / Р. Костер. - ДМК-Пресс, 2018. - с.85-106.

23. Методичні рекомендації до виконання кваліфікаційних робіт здобувачами другого (магістерського) рівня вищої освіти спеціальностей 121 «Інженерія програмного забезпечення» та 122 «Комп’ютерні науки» / Б.І. Мороз, О.В. Іванченко, О.В. Реута, О.С. Шевцова; М-во освіти і науки України, Нац. техн. ун-т “Дніпровська політехніка”. – Дніпро : НТУ «ДП», 2021.

Додаток А

КОД ПРОГРАМИ

DiplomaMag_Character.cpp

```
#include "DiplomaMag_Character.h"
#include "HeadMountedDisplayFunctionLibrary.h"
#include "Camera/CameraComponent.h"
#include "Components/CapsuleComponent.h"
#include "Components/InputComponent.h"
#include "GameFramework/CharacterMovementComponent.h"
#include "GameFramework/Controller.h"
#include "GameFramework/SpringArmComponent.h"

//////////////////////////////////////////////////////////////////////////
// ADiplomaMag_Character

ADiplomaMag_Character::ADiplomaMag_Character()
{
	// Set size for collision capsule
	GetCapsuleComponent()->InitCapsuleSize(42.f, 96.0f);
	// set our turn rates for input
	BaseTurnRate = 45.f;
	BaseLookUpRate = 45.f;
	// Don't rotate when the controller rotates. Let that just affect the camera.
	bUseControllerRotationPitch = false;
	bUseControllerRotationYaw = false;
	bUseControllerRotationRoll = false;

	// Configure character movement
	GetCharacterMovement()->bOrientRotationToMovement = true; // Character moves in the
direction of input...
	GetCharacterMovement()->RotationRate = FRotator(0.0f, 540.0f, 0.0f); // ...at this rotation
rate
	GetCharacterMovement()->JumpZVelocity = 600.f;
	GetCharacterMovement()->AirControl = 0.2f;
	// Create a camera boom (pulls in towards the player if there is a collision)
	CameraBoom = CreateDefaultSubobject<USpringArmComponent>(TEXT("CameraBoom"));
	CameraBoom->SetupAttachment(RootComponent);
	CameraBoom->TargetArmLength = 300.0f; // The camera follows at this distance behind the
character
	CameraBoom->bUsePawnControlRotation = true; // Rotate the arm based on the controller
	// Create a follow camera
	FollowCamera = CreateDefaultSubobject<UCameraComponent>(TEXT("FollowCamera"));
	FollowCamera->SetupAttachment(CameraBoom, USpringArmComponent::SocketName); // Attach the
camera to the end of the boom and let the boom adjust to match the controller orientation
	FollowCamera->bUsePawnControlRotation = false; // Camera does not rotate relative to arm
	// Note: The skeletal mesh and anim blueprint references on the Mesh component (inherited
from Character)
	// are set in the derived blueprint asset named MyCharacter (to avoid direct content
references in C++)
}
//////////////////////////////////////////////////////////////////////////
// Input
void ADiplomaMag_Character::SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent)
{
	// Set up gameplay key bindings
	check(PlayerInputComponent);
	PlayerInputComponent->BindAction("Jump", IE_Pressed, this, &ACharacter::Jump);
	PlayerInputComponent->BindAction("Jump", IE_Released, this, &ACharacter::StopJumping);
	PlayerInputComponent->BindAxis("MoveForward", this, &ADiplomaMag_Character::MoveForward);
	PlayerInputComponent->BindAxis("MoveRight", this, &ADiplomaMag_Character::MoveRight);
	// We have 2 versions of the rotation bindings to handle different kinds of devices
differently
	// "turn" handles devices that provide an absolute delta, such as a mouse.
	// "turnrate" is for devices that we choose to treat as a rate of change, such as an analog
joystick
	PlayerInputComponent->BindAxis("Turn", this, &APawn::AddControllerYawInput);
	PlayerInputComponent->BindAxis("TurnRate", this, &ADiplomaMag_Character::TurnAtRate);
	PlayerInputComponent->BindAxis("LookUp", this, &APawn::AddControllerPitchInput);
	PlayerInputComponent->BindAxis("LookUpRate", this, &ADiplomaMag_Character::LookUpAtRate);
	// handle touch devices
	PlayerInputComponent->BindTouch(IE_Pressed, this, &ADiplomaMag_Character::TouchStarted);
	PlayerInputComponent->BindTouch(IE_Released, this, &ADiplomaMag_Character::TouchStopped);

	// VR headset functionality
	PlayerInputComponent->BindAction("ResetVR", IE_Pressed, this,
&ADiplomaMag_Character::OnResetVR);
```

```
}

void ADiplomaMag_Character::OnResetVR()
{
	// If DiplomaMag_ is added to a project via 'Add Feature' in the Unreal Editor the
dependency on HeadMountedDisplay in DiplomaMag_.Build.cs is not automatically propagated
	// and a linker error will result.
	// You will need to either:
	//		Add "HeadMountedDisplay" to [YourProject].Build.cs
PublicDependencyModuleNames in order to build successfully (appropriate if supporting VR).
	// or:
	//		Comment or delete the call to ResetOrientationAndPosition below (appropriate
if not supporting VR)
	UHeadMountedDisplayFunctionLibrary::ResetOrientationAndPosition();
}

void ADiplomaMag_Character::TouchStarted(ETouchIndex::Type FingerIndex, FVector Location)
{
		Jump();
}

void ADiplomaMag_Character::TouchStopped(ETouchIndex::Type FingerIndex, FVector Location)
{
		StopJumping();
}

void ADiplomaMag_Character::TurnAtRate(float Rate)
{
	// calculate delta for this frame from the rate information
	AddControllerYawInput(Rate * BaseTurnRate * GetWorld()->GetDeltaSeconds());
}

void ADiplomaMag_Character::LookUpAtRate(float Rate)
{
	// calculate delta for this frame from the rate information
	AddControllerPitchInput(Rate * BaseLookUpRate * GetWorld()->GetDeltaSeconds());
}

void ADiplomaMag_Character::MoveForward(float Value)
{
	if ((Controller != nullptr) && (Value != 0.0f))
	{
		// find out which way is forward
		const FRotator Rotation = Controller->GetControlRotation();
		const FRotator YawRotation(0, Rotation.Yaw, 0);

		// get forward vector
		const FVector Direction = FRotationMatrix(YawRotation).GetUnitAxis(EAxis::X);
		AddMovementInput(Direction, Value);
	}
}

void ADiplomaMag_Character::MoveRight(float Value)
{
	if ( (Controller != nullptr) && (Value != 0.0f) )
	{
		// find out which way is right
		const FRotator Rotation = Controller->GetControlRotation();
		const FRotator YawRotation(0, Rotation.Yaw, 0);

		// get right vector
		const FVector Direction = FRotationMatrix(YawRotation).GetUnitAxis(EAxis::Y);
		// add movement in that direction
		AddMovementInput(Direction, Value);
	}
}
```

DiplomaMag_Character.h

```
#pragma once
#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "DiplomaMag_Character.generated.h"

UCLASS(config=Game)
class ADiplomaMag_Character : public ACharacter
{
	GENERATED_BODY()
	/** Camera boom positioning the camera behind the character */
```

```
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera, meta = (AllowPrivateAccess
= "true"))
	class USpringArmComponent* CameraBoom;
	/** Follow camera */
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera, meta = (AllowPrivateAccess
= "true"))
	class UCameraComponent* FollowCamera;
public:
	ADiplomaMag_Character();
	/** Base turn rate, in deg/sec. Other scaling may affect final turn rate. */
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category=Camera)
	float BaseTurnRate;
	/** Base look up/down rate, in deg/sec. Other scaling may affect final rate. */
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category=Camera)
	float BaseLookUpRate;
protected:

	/** Resets HMD orientation in VR. */
	void OnResetVR();
	/** Called for forwards/backward input */
	void MoveForward(float Value);
	/** Called for side to side input */
	void MoveRight(float Value);
	/**
	 * Called via input to turn at a given rate.
	 * @param Rate This is a normalized rate, i.e. 1.0 means 100% of desired turn rate
	 */
	void TurnAtRate(float Rate);

	/**
	 * Called via input to turn look up/down at a given rate.
	 * @param Rate This is a normalized rate, i.e. 1.0 means 100% of desired turn rate
	 */
	void LookUpAtRate(float Rate);

	/** Handler for when a touch input begins. */
	void TouchStarted(ETouchIndex::Type FingerIndex, FVector Location);

	/** Handler for when a touch input stops. */
	void TouchStopped(ETouchIndex::Type FingerIndex, FVector Location);

protected:
	// APawn interface
	virtual void SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent)
override;
	// End of APawn interface

public:
	/** Returns CameraBoom subobject **/
	FORCEINLINE class USpringArmComponent* GetCameraBoom() const { return CameraBoom; }
	/** Returns FollowCamera subobject **/
	FORCEINLINE class UCameraComponent* GetFollowCamera() const { return FollowCamera; }
};
```

AddToProjectConfig.h

```
#pragma once
#include "CoreMinimal.h"
#include "UObject/Object.h"
#include "UObject/Interface.h"
class IClassViewerFilter;
class SWindow;
struct FSlateBrush;
/**
 * Delegate called when code is added to the project.  Passes in the created class name and class
path
 *
 * @param ClassName           The created class name
 * @param ClassPath           The created class path
 * @param ModuleName   The name of the module that the class was added to
 */
DECLARE_DELEGATE_ThreeParams( FOnAddedToProject, const FString& /*ClassName*/, const FString&
/*ClassPath*/, const FString& /*ModuleName*/);
class IClassViewerFilter;
/** Information used when creating a new class via AddCodeToProject */
struct FNewClassInfo
{
	/** The type of class we want to create */
	enum class EClassType : uint8
	{
```

```
		/** The new class is using a UObject as a base, consult BaseClass for the type. */
		UObject,
		/** The new class should be an empty standard C++ class. */
		EmptyCpp,
		/** The new class should be a Slate widget, deriving from SCompoundWidget. */
		SlateWidget,
		/** The new class should be a Slate widget style, deriving from FSlateWidgetStyle,
along with its associated UObject wrapper class. */
		SlateWidgetStyle,
		/** The new class is a UObject Interface, to be implemented by other UObject-based
classes. */
		UInterface,
	};

	/** Default constructor; must produce an object which fails the IsSet check */
	FNewClassInfo()
		: ClassType(EClassType::UObject)
		, BaseClass(nullptr)
	{
	}
	/** Convenience constructor so you can construct from a EClassType */
	explicit FNewClassInfo(const EClassType InClassType)
		: ClassType(InClassType)
		, BaseClass(nullptr)
	{
	}
	/** Convenience constructor so you can construct from a UClass */
	explicit FNewClassInfo(const UClass* InBaseClass)
		: ClassType(EClassType::UObject)
		, BaseClass(InBaseClass)
	{
	}
	/** Check to see if this struct is set to something that could be used to create a new class
*/
	bool IsSet() const
	{
		return ClassType != EClassType::UObject || BaseClass;
	}
	/** Get the "friendly" class name to use in the UI */
	FText GetClassName() const;

	/** Get the class description to use in the UI */
	FText GetClassDescription(const bool bFullDescription = true) const;

	/** Get the class icon to use in the UI */
	const FSlateBrush* GetClassIcon() const;
	/** Get the C++ prefix used for this class type */
	FString GetClassPrefixCPP() const;
	/** Get the C++ class name; this may or may not be prefixed, but will always produce a valid
C++ name via GetClassPrefix() + GetClassName() */
	FString GetClassNameCPP() const;
	/** Some classes may apply a particular suffix; this function returns the class name with
those suffixes removed */
	FString GetCleanClassName(const FString& ClassName) const;
	/** Some classes may apply a particular suffix; this function returns the class name that
will ultimately be used should that happen */
	FString GetFinalClassName(const FString& ClassName) const;
	/** Get the path needed to include this class into another file */
	bool GetIncludePath(FString& OutIncludePath) const;
	/** Gets header filename of the base class. */
	FString GetBaseClassHeaderFilename() const;
	/** Given a class name, generate the header file (.h) that should be used for this class */
	FString GetHeaderFilename(const FString& ClassName) const;
	/** Given a class name, generate the source file (.cpp) that should be used for this class
*/
	FString GetSourceFilename(const FString& ClassName) const;
	/** Get the generation template filename to used based on the current class type */
	FString GetHeaderTemplateFilename() const;
	/** Get the generation template filename to used based on the current class type */
	FString GetSourceTemplateFilename() const;
	/** The type of class we want to create */
	EClassType ClassType;
	/** Base class information; if the ClassType is UObject */
	const UClass* BaseClass;
};
/** Helper that creates lists of featured classes. Include FeaturedClasses.inl for definitions. */
/** @todo make this ini configurable */
struct FFeaturedClasses
{
```

```
public:
	/** Get a list of all featured native class types */
	static TArray<FNewClassInfo> AllNativeClasses();

	/** Get a list of all featured Actor class types */
	static TArray<FNewClassInfo> ActorClasses();

	/** Get a list of all featured Component class types */
	static TArray<FNewClassInfo> ComponentClasses();

private:
	/** Append the featured Actor class types that are commonly used */
	static void AddCommonActorClasses(TArray<FNewClassInfo>& Array);

	/** Append the featured Actor class types that are less commonly used */
	static void AddExtraActorClasses(TArray<FNewClassInfo>& Array);

	/** Append the featured Component class types that are commonly used */
	static void AddCommonComponentClasses(TArray<FNewClassInfo>& Array);

	/** Append the featured Component class types that are less commonly used */
	static void AddExtraComponentClasses(TArray<FNewClassInfo>& Array);
};

/** Add to project dialog configuration structure */
struct FAddToProjectConfig
{
	FAddToProjectConfig() : _ParentClass(nullptr), _bModal(false)	{}

public:
	/** Force the add to project dialog to use the specified parent class. Skips the first step
(choose a parent class) as a result. */
	FAddToProjectConfig& ParentClass(const UClass* InClass)
		{ _ParentClass = InClass; return *this; }
	/** Limits the allowable parent classes by the specified filter. */
	FAddToProjectConfig& AllowableParents(const TSharedPtr<IClassViewerFilter>& In)	{
_ParentClass = nullptr; _AllowableParents = In; return *this; }
	/** The initial path we should use as the destination for the new file, or an empty string
to choose a suitable default based upon the module path. */
	FAddToProjectConfig& InitialPath(FString InInitialPath)
		{ _InitialPath = MoveTemp(InInitialPath); return *this; }
	/** Optional argument that specifies the default name for the new class being added.  The
user will be able to type their own name if they don't like this name.  If empty, defaults to the
name of the inherited class. */
	FAddToProjectConfig& DefaultClassName(FString InDefaultClassName)	{
_DefaultClassName = MoveTemp(InDefaultClassName); return *this; }
	/** Optional argument that specifies the prefix for the new class name.  The user will be
able to type their own name if they don't like this name.  Defaults to "My" if not specified or
empty. */
	FAddToProjectConfig& DefaultClassPrefix(FString InDefaultClassPrefix)	{
_DefaultClassPrefix = MoveTemp(InDefaultClassPrefix); return *this; }
	/** The title text to display on the window */
	FAddToProjectConfig& WindowTitle(FText InText)
		{ _WindowTitle = MoveTemp(InText); return *this; }
	/** The parent window the dialog should use, or null to choose a suitable default parent
window (the main frame, if available). */
	FAddToProjectConfig& ParentWindow(const TSharedPtr<SWindow>& InParentWindow)	{
_ParentWindow = InParentWindow; return *this; }
	/** Make the window modal to force the user to make a decision before continuing. */
	FAddToProjectConfig& Modal(bool bModal = true)
		{ _bModal = bModal; return *this; }
	/** Callback for when the object is successfully added to the project */
	FAddToProjectConfig& OnAddedToProject(const FOnAddedToProject& InDelegate)	{
_OnAddedToProject = InDelegate; return *this; }

	/** Set the add to project dialog to show component types on the initial 'featured' classes
list */
	FAddToProjectConfig& FeatureAllNativeClasses()
		{ _FeaturedClasses = FFeaturedClasses::AllNativeClasses(); return *this; }
	/** Set the add to project dialog to show component types on the initial 'featured' classes
list */
	FAddToProjectConfig& FeatureActorClasses()
		{ _FeaturedClasses = FFeaturedClasses::ActorClasses(); return *this; }
	/** Set the add to project dialog to show component types on the initial 'featured' classes
list */
	FAddToProjectConfig& FeatureComponentClasses()
		{ _FeaturedClasses = FFeaturedClasses::ComponentClasses(); return *this; }
public:
```

```
        /** Forced parent class to use */
        const UClass* _ParentClass;
        /** Filter for allowable parent classes, when ParentClass is nullptr */
        TSharedPtr<IClassViewerFilter> _AllowableParents;
        /** Array of featured classes */
        TArray<FNewClassInfo> _FeaturedClasses;

        /** Initial file path for the (blueprint) class */
        FString _InitialPath;
        /** Default name prefix for the (blueprint) class */
        FString _DefaultClassPrefix;
        /** Default name for the (blueprint) class, excluding class prefix */
        FString _DefaultClassName;

        /** The title to display on the window */
        FText _WindowTitle;
        /** Parent window to use */
        TSharedPtr<SWindow> _ParentWindow;
        /** True to force a modal dialog, false otherwise */
        bool _bModal;

        /** Delegate to invoke when the (blueprint) class has been added to the project */
        FOnAddedToProject _OnAddedToProject;
};
```

ClassTemplateEditorSubsystem.h

```
#pragma once
#include "CoreMinimal.h"
#include "EditorSubsystem.h"
#include "UObject/Class.h"
#include "UObject/WeakObjectPtrTemplates.h"
#include "ClassTemplateEditorSubsystem.generated.h"
// Forward Declarations
class FSubsystemCollectionBase;
class FText;
UCLASS(Abstract)
class GAMEPROJECTGENERATION_API UClassTemplate : public UObject
{
        GENERATED_BODY()

public:
        virtual void BeginDestroy() override;
        // Returns the directory containing the text file template for
        // the given base generated class.
        virtual FString GetDirectory() const;
        // Reads the header template text from disk.  If failure to read from
        // disk, returns false and provides text reason.
        bool ReadHeader(FString& OutHeaderFileText, FText& OutFailReason) const;
        // Reads the source template text from disk.  If failure to read from
        // disk, returns false and provides text reason.
        bool ReadSource(FString& OutSourceFileText, FText& OutFailReason) const;
        const UClass* GetGeneratedBaseClass() const;
protected:
        // Sets the generated base class associated with the given template
        void SetGeneratedBaseClass(UClass* InClass);
        // Returns the filename associated with the provided class template
        // without an extension.  Defaults to class name.
        virtual FString GetFilename() const;
        // Returns full header filename including '.h.template' extension
        FString GetHeaderFilename() const;

        // Returns full sourcefilename including '.cpp.template' extension
        FString GetSourceFilename() const;

private:
        // Base UClass of which template class corresponds.
        UPROPERTY(Transient)
        UClass* GeneratedBaseClass;
};
UCLASS(Abstract)
class GAMEPROJECTGENERATION_API UPluginClassTemplate : public UClassTemplate
{
        GENERATED_BODY()
public:
        // Returns the directory containing the text file template for
        // the given base generated class.
        virtual FString GetDirectory() const override;
protected:
        UPROPERTY(Transient)
```

```
	FString PluginName;
};
UCLASS()
class GAMEPROJECTGENERATION_API UClassTemplateEditorSubsystem : public UEditorSubsystem
{
	GENERATED_BODY()
private:
	using FTemplateRegistry = TMap<TWeakObjectPtr<const UClass>, TWeakObjectPtr<const
UClassTemplate>>;
	FTemplateRegistry TemplateRegistry;
public:
	UClassTemplateEditorSubsystem();

	virtual void Initialize(FSubsystemCollectionBase& Collection) override;
	virtual void Deinitialize() override;
	// Registers all currently loaded template classes with the internal registry.
	void RegisterTemplates();
	// Returns path to the directory containing all engine class templates.
	static FString GetEngineTemplateDirectory();
	// Returns whether or not class has registered template
	bool ContainsClassTemplate(const UClass* InClass) const;
	// Returns class template if one is registered.
	const UClassTemplate* FindClassTemplate(const UClass* InClass) const;
	friend class UClassTemplate;

private:
	// Registers a template class with the subsystem
	void Register(const UClassTemplate* InClassTemplate);
	// Unregisters a template class with the subsystem
	bool Unregister(const UClassTemplate* InClassTemplate);
};
```

GameProjectGenerationModule.h

```
#pragma once
#include "CoreMinimal.h"
#include "Modules/ModuleInterface.h"
#include "Modules/ModuleManager.h"
#include "ModuleDescriptor.h"
#include "AddToProjectConfig.h"
struct FSlateBrush;
struct FTemplateCategory;
/**
 * Game Project Generation module
 */
class FGameProjectGenerationModule : public IModuleInterface
{
public:
	typedef TMap<FName, TSharedPtr<FTemplateCategory>> FTemplateCategoryMap;
	/**
	 * Called right after the plugin DLL has been loaded and the plugin object has been created
	 */
	virtual void StartupModule();
	/**
	 * Called before the plugin is unloaded, right before the plugin object is destroyed.
	 */
	virtual void ShutdownModule();

	/**
	 * Singleton-like access to this module's interface.  This is just for convenience!
	 * Beware of calling this during the shutdown phase, though.  Your module might have been
unloaded already.
	 *
	 * @return Returns singleton instance, loading the module on demand if needed
	 */
	static inline FGameProjectGenerationModule& Get()
	{
		static const FName ModuleName = "GameProjectGeneration";
		return FModuleManager::LoadModuleChecked< FGameProjectGenerationModule >( ModuleName
);
	}
	/** Creates the game project dialog */
	virtual TSharedRef<class SWidget> CreateGameProjectDialog(bool bAllowProjectOpening, bool
bAllowProjectCreate);
	/** Creates a new class dialog for creating classes based on the passed-in class. */
	virtual TSharedRef<class SWidget> CreateNewClassDialog(const UClass* InClass);
	/**
	 * Opens a dialog to add code files to the current project.
	 *
	 * @param	Config		Dialog configuration options
	 */
```

```
        virtual void OpenAddCodeToProjectDialog(const FAddToProjectConfig& Config =
FAddToProjectConfig());
        /**
         * Opens a dialog to add a new blueprint to the current project.
         *
         * @param      Config         Dialog configuration options
         */
        virtual void OpenAddBlueprintToProjectDialog(const FAddToProjectConfig& Config);
        /** Delegate for when the AddCodeToProject dialog is opened */
        DECLARE_EVENT(FGameProjectGenerationModule, FAddCodeToProjectDialogOpenedEvent);
        FAddCodeToProjectDialogOpenedEvent& OnAddCodeToProjectDialogOpened() { return
AddCodeToProjectDialogOpenedEvent; }
        /** Tries to make the project file writable. Prompts to check out as necessary. */
        virtual void TryMakeProjectFileWriteable(const FString& ProjectFile);
        /** Prompts the user to update his project file, if necessary. */
        virtual void CheckForOutOfDateGameProjectFile();
        /** Updates the currently loaded project. Returns true if the project was updated
successfully or if no update was needed */
        virtual bool UpdateGameProject(const FString& ProjectFile, const FString& EngineIdentifier,
FText& OutFailReason);
        /** Updates the current code project */
        virtual bool UpdateCodeProject(FText& OutFailReason, FText& OutFailLog);
        /** Gets the current projects source file count */
        virtual bool ProjectHasCodeFiles();
        /** Returns the path to the module's include header */
        virtual FString DetermineModuleIncludePath(const FModuleContextInfo& ModuleInfo, const
FString& FileRelativeTo);
        /** Get the information about any modules referenced in the .uproject file of the currently
loaded project */
        virtual const TArray<FModuleContextInfo>& GetCurrentProjectModules();
        /** Returns true if the specified class is a valid base class for the given module */
        virtual bool IsValidBaseClassForCreation(const UClass* InClass, const FModuleContextInfo&
InModuleInfo);
        /** Returns true if the specified class is a valid base class for any of the given modules
*/
        virtual bool IsValidBaseClassForCreation(const UClass* InClass, const
TArray<FModuleContextInfo>& InModuleInfoArray);
        /** Gets file and size info about the source directory */
        virtual void GetProjectSourceDirectoryInfo(int32& OutNumFiles, int64& OutDirectorySize);
        /** Warn the user if the project filename is invalid in case they renamed it outside the
editor */
        virtual void CheckAndWarnProjectFilenameValid();
        /** Generate basic project source code */
        virtual bool GenerateBasicSourceCode(TArray<FString>& OutCreatedFiles, FText&
OutFailReason);
        /**
         * Update the list of supported target platforms based upon the parameters provided
         * This will take care of checking out and saving the updated .uproject file automatically
         *
         * @param      InPlatformName        Name of the platform to target (eg, WindowsNoEditor)
         * @param      bIsSupported          true if the platform should be supported by this
project, false if it should not
         */
        virtual void UpdateSupportedTargetPlatforms(const FName& InPlatformName, const bool
bIsSupported);

        /** Clear the list of supported target platforms */
        virtual void ClearSupportedTargetPlatforms();

public:
        // Non DLL-exposed access to template categories
        TSharedPtr<const FTemplateCategory> GetCategory(FName Type) const { return
TemplateCategories.FindRef(Type); }
        void GetAllTemplateCategories(TArray<TSharedPtr<FTemplateCategory>>& OutCategories) const {
TemplateCategories.GenerateValueArray(OutCategories); }
private:
        FAddCodeToProjectDialogOpenedEvent AddCodeToProjectDialogOpenedEvent;
        /** Map of template categories from type to ptr */
        FTemplateCategoryMap TemplateCategories;
        void LoadTemplateCategories();
};
```

GameProjectUtils.h

```
#pragma once
#include "CoreMinimal.h"
#include "SlateFwd.h"
#include "AddToProjectConfig.h"
#include "GameProjectGenerationModule.h"
#include "HardwareTargetingSettings.h"
class UTemplateProjectDefs;
```

```
class UTemplateCategories;
struct FProjectDescriptor;
enum class EClassDomain : uint8;
struct FTemplateConfigValue;
struct FProjectInformation
{
  FProjectInformation() = default;

  FString ProjectFilename;
  FString TemplateFile;
  FName TemplateCategory;
  bool bShouldGenerateCode = false;
  bool bCopyStarterContent = false;
  bool bIsBlankTemplate = false;
  bool bIsEnterpriseProject = false;
  bool bForceExtendedLuminanceRange; // See
"r.DefaultFeature.AutoExposure.ExtendDefaultLuminanceRange"
  // These are all optional, because there is an additional state introduced by hiding the setting
in the template.
  // In this case, the template author has chosen not to give the user a choice,
  // so we must assume that the template already controls these settings explicitly.
  TOptional<bool> bEnableXR;
  TOptional<bool> bEnableRaytracing;
  TOptional<EHardwareClass::Type> TargetedHardware;
  TOptional<EGraphicsPreset::Type> DefaultGraphicsPerformance;
  /** The name of the feature pack to use as starter content. Must be located under FeaturePacks\.
*/
  FString StarterContent;
};
DECLARE_DELEGATE_RetVal_OneParam(bool, FProjectDescriptorModifier, FProjectDescriptor&);
class GAMEPROJECTGENERATION_API GameProjectUtils
{
public:
  /** Where is this class located within the Source folder? */
  enum class EClassLocation : uint8
  {
     /** The class is going to a user defined location (outside of the Public, Private, or Classes)
folder for this module */
     UserDefined,

     /** The class is going to the Public folder for this module */
     Public,

     /** The class is going to the Private folder for this module */
     Private,

     /** The class is going to the Classes folder for this module */
     Classes,
  };
  /** Used as a function return result when adding new code to the project */
  enum class EAddCodeToProjectResult : uint8
  {
     /** Function has successfully added the code and hot-reloaded the required module(s) */
     Succeeded,
     /** There were errors with the input given to the function */
     InvalidInput,
     /** There were errors when adding the new source files */
     FailedToAddCode,
     /** There were errors when hot-reloading the new module */
     FailedToHotReload,
  };
  /** Used as a function return result when a project is duplicated when upgrading project's version
in Convert project dialog - Open a copy */
  enum class EProjectDuplicateResult : uint8
  {
     /** Function has successfully duplicated all project files */
     Succeeded,
     /** There were errors while duplicating project files */
     Failed,
     /** User has canceled project duplication process */
     UserCanceled
  };
  /** Returns true if the project filename is properly formed and does not conflict with another
project */
  static bool IsValidProjectFileForCreation(const FString& ProjectFile, FText& OutFailReason);
  /** Opens the specified project, if it exists. Returns true if the project file is valid. On
failure, OutFailReason will be populated. */
  static bool OpenProject(const FString& ProjectFile, FText& OutFailReason);
```

```
  /** Opens the code editing IDE for the specified project, if it exists. Returns true if the IDE
could be opened. On failure, OutFailReason will be populated. */
  static bool OpenCodeIDE(const FString& ProjectFile, FText& OutFailReason);
  /** Creates the specified project file and all required folders. If TemplateFile is non-empty, it
will be used as the template for creation. On failure, OutFailReason will be populated. */
  static bool CreateProject(const FProjectInformation& InProjectInfo, FText& OutFailReason, FText&
OutFailLog, TArray<FString>* OutCreatedFiles = nullptr);
  /** Prompts the user to update his project file, if necessary. */
  static void CheckForOutOfDateGameProjectFile();
  /** Warn the user if the project filename is invalid in case they renamed it outside the editor */
  static void CheckAndWarnProjectFilenameValid();
  /** Checks out the current project file (or prompts to make writable) */
  static void TryMakeProjectFileWriteable(const FString& ProjectFile);
  /** Updates the given project file to an engine identifier. Returns true if the project was
updated successfully or if no update was needed */
  static bool UpdateGameProject(const FString& ProjectFile, const FString& EngineIdentifier, FText&
OutFailReason);
  /**
   * Opens a dialog to add code files or blueprints to the current project.
   *
   * @param     Config                Configuration options for the dialog
   * @param     InDomain              The domain of the class we're creating (native or blueprint)
   */
  static void OpenAddToProjectDialog(const FAddToProjectConfig& Config, EClassDomain InDomain);
  /** Returns true if the specified class name is properly formed and does not conflict with another
class */
  static bool IsValidClassNameForCreation(const FString& NewClassName, FText& OutFailReason);
  /** Returns true if the specified class name is properly formed and does not conflict with another
class, including source/header files */
  static bool IsValidClassNameForCreation(const FString& NewClassName, const FModuleContextInfo&
ModuleInfo, const TSet<FString>& DisallowedHeaderNames, FText& OutFailReason);
  /** Returns true if the specified class is a valid base class for the given module */
  static bool IsValidBaseClassForCreation(const UClass* InClass, const FModuleContextInfo&
InModuleInfo);
  /** Returns true if the specified class is a valid base class for any of the given modules */
static bool IsValidBaseClassForCreation(const UClass* InClass, const TArray<FModuleContextInfo>&
InModuleInfoArray);
  /** Adds new source code to the project. When returning Succeeded or FailedToHotReload,
OutSyncFileAndLineNumber will be the the preferred target file to sync in the users code editing
IDE, formatted for use with GenericApplication::GotoLineInSource */
  static EAddCodeToProjectResult AddCodeToProject(const FString& NewClassName, const FString&
NewClassPath, const FModuleContextInfo& ModuleInfo, const FNewClassInfo ParentClassInfo, const
TSet<FString>& DisallowedHeaderNames, FString& OutHeaderFilePath, FString& OutCppFilePath, FText&
OutFailReason);
  /** Loads a list of template categories defined in the TemplateCategories.ini file in the
specified folder */
  static UTemplateCategories* LoadTemplateCategories(const FString& RootDir);
  /** Loads a template project definitions object from the TemplateDefs.ini file in the specified
project */
  static UTemplateProjectDefs* LoadTemplateDefs(const FString& ProjectDirectory);
  /** @return The number of code files in the currently loaded project */
  static int32 GetProjectCodeFileCount();
  /**
  * Retrieves file and size info about the project's source directory
  * @param OutNumFiles Contains the number of files within the source directory
  * @param OutDirectorySize Contains the combined size of all files in the directory
  */
  static void GetProjectSourceDirectoryInfo(int32& OutNumFiles, int64& OutDirectorySize);
  /** Returns the uproject template filename for the default project template. */
  static FString GetDefaultProjectTemplateFilename();
  /** Compiles a project while showing a progress bar, and offers to open the IDE if it fails. */
  static bool BuildCodeProject(const FString& ProjectFilename);

  /** Creates code project files for a new game project. On failure, OutFailReason and OutFailLog
will be populated. */
  static bool GenerateCodeProjectFiles(const FString& ProjectFilename, FText& OutFailReason, FText&
OutFailLog);
  /** Returns true if there are starter content files available for instancing into new projects. */
  static bool IsStarterContentAvailableForNewProjects();
  static bool IsStarterContentAvailableForProject(const FProjectInformation& ProjectInfo);
  /**
   * Get the information about any modules referenced in the .uproject file of the currently loaded
project
   */
  static const TArray<FModuleContextInfo>& GetCurrentProjectModules();
  /**
  * Get the information about any modules in any of the plugins in the currently loaded project
(Ignores Engine Plugins)
  */
```

```
  static TArray<FModuleContextInfo> GetCurrentProjectPluginModules();

  /**
   * Reset the cached result of information about any modules referenced in the .uproject file of
the currently loaded project
   */
  static void ResetCurrentProjectModulesCache();

  /**
   * Check to see if the given path is a valid place to put source code for this project (exists
within the source root path)
   *
   * @param     InPath                        The path to check
   * @param     ModuleInfo                    Information about the module being validated
   * @param     OutFailReason         Optional parameter to fill with failure information
   *
   * @return    true if the path is valid, false otherwise
   */
  static bool IsValidSourcePath(const FString& InPath, const FModuleContextInfo& ModuleInfo, FText*
const OutFailReason = nullptr);
  /**
   * Given the path provided, work out where generated .h and .cpp files would be placed
   *
   * @param     InPath                        The path to use a base
   * @param     ModuleInfo                    Information about the module being validated
   * @param     OutHeaderPath         The path where the .h file should be placed
   * @param     OutSourcePath         The path where the .cpp file should be placed
   * @param     OutFailReason         Optional parameter to fill with failure information
   *
   * @return    false if the paths are invalid
   */
  static bool CalculateSourcePaths(const FString& InPath, const FModuleContextInfo& ModuleInfo,
FString& OutHeaderPath, FString& OutSourcePath, FText* const OutFailReason = nullptr);
  /**
   * Given the path provided, work out where it's located within the Source folder
   *
   * @param     InPath                        The path to use a base
   * @param     ModuleInfo                    Information about the module being validated
   * @param     OutClassLocation      The location within the Source folder
   * @param     OutFailReason         Optional parameter to fill with failure information
   *
   * @return    false if the paths are invalid
   */
  static bool GetClassLocation(const FString& InPath, const FModuleContextInfo& ModuleInfo,
EClassLocation& OutClassLocation, FText* const OutFailReason = nullptr);
  /** Creates a copy of a project directory in order to upgrade it. */
  static EProjectDuplicateResult DuplicateProjectForUpgrade( const FString& InProjectFile, FString&
OutNewProjectFile );
  /**
   * Update the list of supported target platforms based upon the parameters provided
   * This will take care of checking out and saving the updated .uproject file automatically
   *
   * @param     InPlatformName        Name of the platform to target (eg, WindowsNoEditor)
   * @param     bIsSupported          true if the platform should be supported by this project,
false if it should not
   */
  static void UpdateSupportedTargetPlatforms(const FName& InPlatformName, const bool bIsSupported);
  /** Clear the list of supported target platforms */
  static void ClearSupportedTargetPlatforms();
  /** Returns the path to the module's include header */
  static FString DetermineModuleIncludePath(const FModuleContextInfo& ModuleInfo, const FString&
FileRelativeTo);
  /** Creates the basic source code for a new project. On failure, OutFailReason will be populated.
*/
  static bool GenerateBasicSourceCode(TArray<FString>& OutCreatedFiles, FText& OutFailReason);
  /** Generates a Build.cs file for a game module */
  static bool GenerateGameModuleBuildFile(const FString& NewBuildFileName, const FString&
ModuleName, const TArray<FString>& PublicDependencyModuleNames, const TArray<FString>&
PrivateDependencyModuleNames, FText& OutFailReason);
  /** Generates a Build.cs file for a plugin module. Set 'bUseExplicitOrSharedPCHs' to false to
disable IWYU conventions. */
  static bool GeneratePluginModuleBuildFile(const FString& NewBuildFileName, const FString&
ModuleName, const TArray<FString>& PublicDependencyModuleNames, const TArray<FString>&
PrivateDependencyModuleNames, FText& OutFailReason, bool bUseExplicitOrSharedPCHs = true);
  /** Generates a module .cpp file, intended for plugin use */
  static bool GeneratePluginModuleCPPFile(const FString& CPPFileName, const FString& ModuleName,
const FString& StartupSourceCode, FText& OutFailReason);
  /** Generates a module .h file, intended for plugin use */
```

```
  static bool GeneratePluginModuleHeaderFile(const FString& HeaderFileName, const TArray<FString>&
PublicHeaderIncludes, FText& OutFailReason);
  /** Returns true if the currently loaded project has code files */
  static bool ProjectHasCodeFiles();
  /** Returns the contents of the specified template file */
  static bool ReadTemplateFile(const FString& TemplateFileName, FString& OutFileContents, FText&
OutFailReason);
  /** Writes an output file. OutputFilename includes a path */
  static bool WriteOutputFile(const FString& OutputFilename, const FString& OutputFileContents,
FText& OutFailReason);
  /** Returns a comma delimited string comprised of all the elements in InList. If
bPlaceQuotesAroundEveryElement, every element is within quotes. */
  static FString MakeCommaDelimitedList(const TArray<FString>& InList, bool
bPlaceQuotesAroundEveryElement = true);
  /** Checks the name for illegal characters */
  static bool NameContainsOnlyLegalCharacters(const FString& TestName, FString&
OutIllegalCharacters);
  /** Returns a list of #include lines formed from InList */
  static FString MakeIncludeList(const TArray<FString>& InList);
  /** Deletes the specified list of files that were created during file creation */
  static void DeleteCreatedFiles(const FString& RootFolder, const TArray<FString>& CreatedFiles);
  /**
   * Update the list of plugin directories to scan
   * This will take care of checking out and saving the updated .uproject file automatically
   *
   * @param     InDir directory to add/remove
   * @param     bAddOrRemove true if the directory should be added to this project, false if it
should not
   * @return Whether the plugin directory list was changed
   */
  static bool UpdateAdditionalPluginDirectory(const FString& InDir, const bool bAddOrRemove);

  /** Gets the default build settings version for UBT */
  static const TCHAR* GetDefaultBuildSettingsVersion();
private:
  /** Add hardware-specific config values such as the target platform and RHI. */
  static void AddHardwareConfigValues(const FProjectInformation& InProjectInfo,
TArray<FTemplateConfigValue>& ConfigValues);
  /** Get the name of the starter content pack to use for the given project. */
  static FString GetStarterContentName(const FProjectInformation& InProjectInfo);
  /** Generates a new project without using a template project */
  static TOptional<FGuid> GenerateProjectFromScratch(const FProjectInformation& InProjectInfo,
FText& OutFailReason, FText& OutFailLog);

  /** Generates a new project using a template project */
  static TOptional<FGuid> CreateProjectFromTemplate(const FProjectInformation& InProjectInfo, FText&
OutFailReason, FText& OutFailLog, TArray<FString>* OutCreatedFiles = nullptr);
  /** Sets the engine association for a new project. Handles foreign and non-foreign projects. */
  static bool SetEngineAssociationForForeignProject(const FString& ProjectFileName, FText&
OutFailReason);
  /** Insert any required feature packs into the DefaultGame.ini file */
  static bool InsertFeaturePacksIntoINIFile(const FProjectInformation& InProjectInfo, FText&
OutFailReason);
  /*
   * Insert the addition files from any feature packs specified in the temapalte defs file
   * @param InProjectInfo             Project infor to add content for
   * @param CreatedFiles              List of files we copied
   * @param OutFailReason             Failure reason (if any)
   *
   * @returns true if no errors
   */
  static bool AddSharedContentToProject(const FProjectInformation &InProjectInfo, TArray<FString>
&CreatedFiles, FText& OutFailReason);
  /** Returns the template defs ini filename */
  static FString GetTemplateDefsFilename();
  /** Returns the include header path for a given fully specified, normalized file path */
  static FString GetIncludePathForFile(const FString& InFullFilePath, const FString&
ModuleRootPath);
  /** Checks the name for an underscore and the existence of XB1 XDK */
  static bool NameContainsUnderscoreAndXB1Installed(const FString& TestName);
  /** Returns true if the project file exists on disk */
  static bool ProjectFileExists(const FString& ProjectFile);
  /** Returns true if any project files exist in the given folder */
  static bool AnyProjectFilesExistInFolder(const FString& Path);
  /** Returns true if file cleanup on failure is enabled, false if not */
  static bool CleanupIsEnabled();
  /** Creates ini files for a new project. On failure, OutFailReason will be populated. */
  static bool GenerateConfigFiles(const FProjectInformation& InProjectInfo, TArray<FString>&
OutCreatedFiles, FText& OutFailReason, FGuid& OutProjectID);
```

```
  /* Creates new ini files for a specific project's platform configurations. */
  static bool GeneratePlatformConfigFiles(const FProjectInformation& InProjectInfo, FText&
OutFailReason);
  /** Creates the basic source code for a new project. On failure, OutFailReason will be populated.
*/
  static bool GenerateBasicSourceCode(const FString& NewProjectSourcePath, const FString&
NewProjectName, const FString& NewProjectRoot, TArray<FString>& OutGeneratedStartupModuleNames,
TArray<FString>& OutCreatedFiles, FText& OutFailReason);
  /** Creates the game framework source code for a new project (Pawn, GameMode, PlayerControlleR).
On failure, OutFailReason will be populated. */
  static bool GenerateGameFrameworkSourceCode(const FString& NewProjectSourcePath, const FString&
NewProjectName, TArray<FString>& OutCreatedFiles, FText& OutFailReason);
  /** Creates the batch file to regenerate code projects. */
  static bool GenerateCodeProjectGenerationBatchFile(const FString& ProjectFolder, TArray<FString>&
OutCreatedFiles, FText& OutFailReason);
  /** Creates the batch file for launching the editor or game */
  static bool GenerateLaunchBatchFile(const FString& ProjectName, const FString& ProjectFolder, bool
bLaunchEditor, TArray<FString>& OutCreatedFiles, FText& OutFailReason);
  /** Returns the copyright line used at the top of all files */
  static FString MakeCopyrightLine();
  /** Generates a header file for a UObject class. OutSyncLocation is a string representing the
preferred cursor sync location for this file after creation. */
  static bool GenerateClassHeaderFile(const FString& NewHeaderFileName, const FString
UnPrefixedClassName, const FNewClassInfo ParentClassInfo, const TArray<FString>& ClassSpecifierList,
const FString& ClassProperties, const FString& ClassFunctionDeclarations, FString& OutSyncLocation,
const FModuleContextInfo& ModuleInfo, bool bDeclareConstructor, FText& OutFailReason);
  /** Finds the cursor sync location in the source file and reports it back as a string */
  static void HarvestCursorSyncLocation( FString& FinalOutput, FString& OutSyncLocation );
  /** Generates a cpp file for a UObject class */
  static bool GenerateClassCPPFile(const FString& NewCPPFileName, const FString UnPrefixedClassName,
const FNewClassInfo ParentClassInfo, const TArray<FString>& AdditionalIncludes, const
TArray<FString>& PropertyOverrides, const FString& AdditionalMemberDefinitions, FString&
OutSyncLocation, const FModuleContextInfo& ModuleInfo, FText& OutFailReason);
  /** Generates a Target.cs file for a game module */
  static bool GenerateGameModuleTargetFile(const FString& NewTargetFileName, const FString&
ModuleName, const TArray<FString>& ExtraModuleNames, FText& OutFailReason);
  /** Generates a Build.cs file for a Editor module */
  static bool GenerateEditorModuleBuildFile(const FString& NewBuildFileName, const FString&
ModuleName, const TArray<FString>& PublicDependencyModuleNames, const TArray<FString>&
PrivateDependencyModuleNames, FText& OutFailReason);

  /** Generates a Target.cs file for a Editor module */
  static bool GenerateEditorModuleTargetFile(const FString& NewTargetFileName, const FString&
ModuleName, const TArray<FString>& ExtraModuleNames, FText& OutFailReason);
  /** Generates a main game module cpp file */
  static bool GenerateGameModuleCPPFile(const FString& NewGameModuleCPPFileName, const FString&
ModuleName, const FString& GameName, FText& OutFailReason);
  /** Generates a main game module header file */
  static bool GenerateGameModuleHeaderFile(const FString& NewGameModuleHeaderFileName, const
TArray<FString>& PublicHeaderIncludes, FText& OutFailReason);
  /**
  * Replace a wildcard with another string
  *
  * @param Input                              The input string
  * @param From                         The wildcard to be replaced
  * @param To                                 The text the wildcard should be replaced with
  * @param bLeadingTab         If the line where the wildcard is located starts with a tab
  * @param bTrailingNewLine    If the line where the wildcard is located ends with a new line

  * @return the input string after replacement of the wildcard
  */
  static FString ReplaceWildcard(const FString& Input, const FString& From, const FString& To, bool
bLeadingTab = false, bool bTrailingNewLine = false);
  /** Handler for when the user confirms a project update */
  static void OnUpdateProjectConfirm();
  /** @param OutProjectCodeFilenames Contains the filenames of the project source code files */
  static void GetProjectCodeFilenames(TArray<FString>& OutProjectCode
  /**
   * Updates the projects and modifies FProjectDescriptor accordingly to given modifier.
   *
   * @param Modifier   Callback delegate that will modify the project descriptor accordingly.
   */
  static void UpdateProject(const FProjectDescriptorModifier& Modifier);

  /**
   * Updates the project file.
   */
  static void UpdateProject();
```

```
  /**
   * Updates the projects, and optionally the modules names
   *
   * @param StartupModuleNames if specified, replaces the existing module names with this version
   */
  static void UpdateProject(const TArray<FString>* StartupModuleNames);
  /** Handler for when the user opts out of a project update */
  static void OnUpdateProjectCancel();
  /**
   * Updates the loaded game project file to the current version and to use the given modules
   *
   * @param ProjectFilename           The name of the project (used to checkout from source control)
   * @param EngineIdentifier          The identifier for the engine to open the project with
   * @param StartupModuleNames if specified, replaces the existing module names with this version
   * @param OutFailReason                     Out, if unsuccessful this is the reason why
   * @return true, if successful
   */
  static bool UpdateGameProjectFile(const FString& ProjectFilename, const FString& EngineIdentifier,
const TArray<FString>* StartupModuleNames, FText& OutFailReason);
  /**
   * Updates the loaded game project file to the current version and modifies FProjectDescriptor
accordingly to given modifier.
   *
   * @param ProjectFilename           The name of the project (used to checkout from source control)
   * @param EngineIdentifier          The identifier for the engine to open the project with
   * @param Modifier                          Callback delegate that will modify the project
descriptor accordingly.
   * @param OutFailReason                     Out, if unsuccessful this is the reason why
   * @return true, if successful
   */
  static bool UpdateGameProjectFile(const FString& ProjectFilename, const FString& EngineIdentifier,
const FProjectDescriptorModifier& Modifier, FText& OutFailReason);
  /**
   * Updates the loaded game project file to the current version.
   *
   * @param ProjectFilename           The name of the project (used to checkout from source control)
   * @param EngineIdentifier          The identifier for the engine to open the project with
   * @param OutFailReason                     Out, if unsuccessful this is the reason why

   * @return true, if successful
   */
  static bool UpdateGameProjectFile(const FString& ProjectFilename, const FString& EngineIdentifier,
FText& OutFailReason);
  /** Checks the specified game project file out from source control */
  static bool CheckoutGameProjectFile(const FString& ProjectFilename, FText& OutFailReason);
  /** Internal handler for AddCodeToProject*/
  static EAddCodeToProjectResult AddCodeToProject_Internal(const FString& NewClassName, const
FString& NewClassPath, const FModuleContextInfo& ModuleInfo, const FNewClassInfo ParentClassInfo,
const TSet<FString>& DisallowedHeaderNames, FString& OutHeaderFilePath, FString& OutCppFilePath,
FText& OutFailReason);
  /** Internal handler for IsValidBaseClassForCreation */
  DECLARE_DELEGATE_RetVal_OneParam(bool, FDoesClassNeedAPIExportCallback, const FString&
/*ClassModuleName*/);
  static bool IsValidBaseClassForCreation_Internal(const UClass* InClass, const
FDoesClassNeedAPIExportCallback& InDoesClassNeedAPIExport);
  /** Handler for the user confirming they've read the name legnth warning */
  static void OnWarningReasonOk();

  /** Given a source file name, find its location within the project */
  static bool FindSourceFileInProject(const FString& InFilename, const FString& InSearchPath,
FString& OutPath);
  /**
   * Gets required additional dependencies for fresh project for given class info.
   *
   * @param ClassInfo Given class info.
   *
   * @returns Array of required dependencies.
   */
  static TArray<FString> GetRequiredAdditionalDependencies(const FNewClassInfo& ClassInfo);
  /**
   * Updates startup module names in project descriptor.
   *
   * @param Descriptor Descriptor to update.
   * @param StartupModuleNames Modules to fill.
   *
   * @returns True if descriptor has been modified. False otherwise.
   */
  static bool UpdateStartupModuleNames(FProjectDescriptor& Descriptor, const TArray<FString>*
StartupModuleNames);
```

```
  /**
   * Updates additional dependencies in project descriptor.
   *
   * @param Descriptor Descriptor to update.
   * @param RequiredDependencies Required dependencies.
   * @param ModuleName Module name for which those dependencies are required.
   *
   * @returns True if descriptor has been modified. False otherwise.
   */
  static bool UpdateRequiredAdditionalDependencies(FProjectDescriptor& Descriptor, TArray<FString>&
RequiredDependencies, const FString& ModuleName);
  /**
   * Updates the projects and modifies FProjectDescriptor accordingly to given modifier.
   *
   * @param Modifier   Callback delegate that will modify the project descriptor accordingly.
   */
  static void UpdateProject_Impl(const FProjectDescriptorModifier* Modifier);
  /**
   * Updates the loaded game project file to the current version and modifies FProjectDescriptor
accordingly to given modifier.
   *
   * @param ProjectFilename           The name of the project (used to checkout from source control)
   * @param EngineIdentifier          The identifier for the engine to open the project with
   * @param Modifier                          Callback delegate that will modify the project
descriptor accordingly.
   * @param OutFailReason                     Out, if unsuccessful this is the reason why

   * @return true, if successful
   */
  static bool UpdateGameProjectFile_Impl(const FString& ProjectFilename, const FString&
EngineIdentifier, const FProjectDescriptorModifier* Modifier, FText& OutFailReason);
private:
  static TWeakPtr<SNotificationItem> UpdateGameProjectNotification;
  static TWeakPtr<SNotificationItem> WarningProjectNameNotification;

  // Whether we should use AudioMixer for all platforms:
  static bool bUseAudioMixerForAllPlatforms;
  constexpr static const TCHAR IncludePathFormatString[] = TEXT("#include \"%s\"");
};
```

Додаток Б

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

Факультет інформаційних технологій
Кафедра програмного забезпечення комп'ютерних систем

ВІДГУК

Керівника економічної частини Професора Вагонової О.Г.

(прізвище, ім'я, по батькові, вчене звання, посада, місце роботи)

На кваліфікаційну роботу ступення магістра

Студента ІІ курсу групи 121м-20-1 Мішина Гліба Денисовича

На тему Дослідження ефективності застосування методів процедурної генерації для просторового орієнтування на основі грального двигуна Unreal Engine

«___»_____2022 р.

(підпис)

Додаток В

ПЕРЕЛІК ДОКУМЕНТІВ НА ДИСКУ

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